

# **THE AMSTRAD CP/M PLUS**

**ANDREW R. M. CLARKE**

**DAVID POWYS-LYBBE**

**AMSTRAD is a registered trademark of  
Amstrad Consumer Electronics PLC  
CP/M Plus is a trademark of Digital Research Inc.**







# THE AMSTRAD CP/M PLUS

by  
Andrew R. M. Clarke and  
David Powys-Lybbe

AMSTRAD is a registered trademark of AMSTRAD Consumer  
Electronics plc.

CP/M Plus is a trademark of Digital Research Inc.



Published by © 1986 M.M.L. Systems Ltd, 11 Sun Street, London EC2M 2PS

First Published April 1986

Second Edition June 1986

This edition of the book was printed and indexed with NEWWORD 3 driving a BDS Laser Printer.

ISBN 1 869910 05 2 (paperback)

ISBN 1 869910 00 1 (ring binder)

A disc, containing the programs printed in this book is available from the publishers.

## Useful Addresses

The Authors:-

MML Systems Ltd,  
11 Sun Street,  
London EC2M 2PS  
Tel: 01-247 0691

Amstrad CP/M Software from:-

NewStar Software Ltd,  
200 North Service Road,  
Brentwood, Essex CM14 4SG  
Tel: 0277 220573  
Telex: 995143 NEWSTAR G  
FAX: 0277 232637

Amstrad:-

Brentwood House,  
169 Kings Road,  
Brentwood, Essex CM14 4EF  
Tel: 0277 230222

CP/M Users Group UK:-

Diana Fordred,  
72 Mill Road,  
Hawley, Dartford, Kent  
Tel: 0322 22669

Digital Research:-

Oxford House,  
Oxford Street,  
Newbury, Berks RG13 1JB  
Tel: 0635 35304



## Preface

The Amstrad CP/M Plus computers are, potentially, very powerful micros with a huge established range of software to run on them. This is due to the presence of the operating system CP/M Plus.

'The Amstrad CP/M Plus' is the book that was originally designed by Amstrad to unlock the power of CP/M Plus for the Amstrad Computer User. It is written to be used by all CP/M Plus users from the raw beginner to the professional programmer, and contains details about CP/M Plus that have never before been published. It can be used as a reference guide but it is intended to be readable too.

Writing books on CP/M is a remarkable cottage industry, and it is with much timidity that the authors embarked on what turned into a huge project as they became more and more determined to make the book the best and the most complete book on CP/M Plus ever published.

The book regards the CP/M phenomenon as being due to more than just an efficient operating system. It is about application software and languages too, and so the book discusses the most popular packages that run under CP/M.

The book is in five main parts. The first part tries to explain what CP/M is, and why it has become so popular. It also gives a historical introduction. The second part is for the CP/M user, and covers all the important CP/M operations. The third part is for the programmer and student of CP/M and describes how to grapple with, and use, the power of CP/M. The fourth part is concerned with the common CP/M languages and assemblers. The final part of the book contains a reference for the CP/M user, and gives details of the implementation on the Amstrad machines. This material complements the information in the manual and is intended as a reference for the more sophisticated user of the machine.

The book starts with a history of CP/M on both 8 and 16-bit micros, and deals with the major software packages such as Multiplan and Wordstar that have contributed to CP/M's success. It also explains the demise of the 8-bit micros as business computers in the wake of the integrated 'super-spreadsheet'. It goes on to clarify the reasons for the resurgence in the popularity of Z80-based computers.

The next section of the book is for the beginner and deals with the various tasks that are necessary to get 'up and running'. It explains what to do to achieve such ends as writing a file or formatting a disc without attempting to explain the reasons. This section is followed by a chapter that describes each CP/M transient command in detail, with examples of usage. In the final part of this section of the book is a chapter on using CP/M for communicating to other computers, connecting to peripherals, and telecommunications.

Following the part of the book dealing with the use of CP/M is a section on writing CP/M software. As CP/M Plus is such a sophisticated operating system, this is of necessity a huge undertaking and this section is a 'book within a book'. To start with there is an introduction describing some of the best CP/M languages, giving general tips, and providing worked examples (written in C) of tasks that require the programmer to relate directly to the BDOS. Following this is the chapter on the BDOS. Just as the BDOS is the heart



## The Preface

of CP/M, so this chapter is the heart of the book and describes the BDOS calls and their usage in a detail never before attempted in any publication. When this chapter was written, every BDOS call was tested and conflicts with the CP/M documentation noted. The result is of use to every CP/M programmer and must lay claim to being a definitive text. Following this chapter is a description of RSXs, along with a worked example of a background spooler (written in C). The section is rounded off with a full description of GSX, the graphics extension, with examples of its usage.

The final section of the book describes various languages in more detail. CBASIC, MBASIC, BDS C, Astec C, Small C, Pascal MT+ and Algol M are all described. BASIC-E, the BASIC that has most claim to being the native CP/M BASIC is given its own appendix. The next chapter describes how to use a relocating macro assembler and linker (RMAC and LINK), and introduces, as an example, an assembly code program that emulates the C 'printf' function to provide a decimal/octal or binary to hexadecimal converter.

There are a number of appendices. The first is an essay on the CP/M assemblers that are available to run under CP/M and a description of the most common pseudo-ops and macros. It also gives all the XITAN/TDL Z80 opcodes. The next appendix describes BASIC-E, the original CP/M BASIC, in detail. A third appendix gives the full details of the AMSTRAD implementation of CP/M Plus operating system and fully describes all the firmware calls. The next appendix covers the AMSTRAD utilities. The final appendix discusses the anatomy of the Console Command Processor as an illustration of the way that CP/M is used to the fullest extent.

This book has been extraordinarily difficult to publish. It was originally commissioned by AMSOFT but was then tackled by a succession of publishers who balked at its length until the authors, in desperation, published it themselves. This book should be treasured as an essential piece of microcomputer memorabilia from the vague period of prehistory when Z80-based machines roamed the swamps of the megalithic aeon.

The authors of this book are enthusiasts for CP/M and are also experienced CP/M system implementers. They are engaged professionally in developing CP/M applications such as communications, financial and business modelling, graphics, and accounting. Andrew Clarke is editor of the Journal of the CP/M User group (UK). David Powys-Lybbe is an acknowledged expert on CP/M who has written widely and has spent time working for Digital Research as Principal Support Consultant.

**William Poel Newstar Software Ltd**

This is the second edition of the book. We acknowledge the help of Amstrad for their help in supplying information and allowing us to reprint the details of the implementation of CP/M Plus on the 6128 and 8256. We also thank Locomotive Software Ltd. Digital Research have also been very helpful and we are grateful to them for this. Some material that we have incorporated has been taken from other sources. Generally, we have no idea of the authorship. Information on BDS C comes from the public-domain documentation written by Lear Zolman, Public-domain material from Gordon E. Eubanks Jr was used for the documentation on BASIC-E. The material on the implementation is the copyright of Locomotive Software Ltd. Mike Skliros helped us enormously with the chapters written for the beginner. As an enthusiastic beginner himself, he was well placed to relieve the pedantry with humour. Finally we must give our grateful thanks to NewStar Software Ltd for their help in lending us their facilities to help to publish this edition.



# The Contents

## ---- Section 1     Introducing CP/M ----

Chapter 1. Page 5  
The history of CP/M. The genesis of the first portable operating system. The evolution of the concept of the BIOS. CP/M 1.3, CP/M 1.4 The arrival of the 5 1/4 in. drive and the idea of the disk parameter table. (CP/M 2.2). MP/M, the multitasking CP/M. The 8088/6 revolution (CP/M-86). MSDOS and the market gap. Concurrent CP/M; the gap widens. The attempt at a graphics operating system (GSX). CP/M+, the state of the art. CP/M goes visual (GEM). The Unix diversion. CP/M-68K.

Chapter 2. Page 10  
CP/M computers. The CP/M-lookalikes. Turbodos, TPM, CDOS, and MSDOS. The effect of hardware standardisation. The major software running under CP/M. Spreadsheets, wordprocessors, and databases. The arrival of integrated software. CP/M and MSDOS.

## ---- Section 2     Using CP/M ----

Chapter 3. Page 15  
Getting Started. How to switch on. How to copy a disk. How to run a program. How to type a file. ..copy a file, ..change disks, ..change the contents of a file, ..delete a file, ..format a disk, etc. etc..

Chapter 4. Page 36  
The CP/M Commands. the Built-in commands and transients. How to use PIP, SID, MAC, ED, DIR, INITDIR, SHOW SET etc.

Chapter 5. Page 99  
Communications using CP/M. Serial ports. Parallel ports. Modems etc. Connecting with other computers. The usefulness of PIP. Bstam, Xmodem, Ascom, Modem7, etc.

## ---- Section 3     Writing CP/M Software ----

Chapter 6 Page 113  
The CP/M languages. How to interface with CP/M from a high level language. How to write well mannered CP/M software. Dos and donts.

Chapter 7 Page 132  
The BDOS functions and their calls. The BIOS functions and their calls.

Chapter 8 Page 264  
Extending the operating system. Why use RSXs? How to write RSXs. Writing a background spooler.

Chapter 9 Page 272  
GSX and how to use it. Device independence in graphics and the GKS interface. Writing portable graphical software.

## The Contents

### ---- Section 4    Running common CP/M software    ----

Chapter 10	Page 323
Using BDS C, Small C and ASTEC C. Using BASIC-E MBASIC and CBASIC. Using Algol-M and Pascal MT+ etc.	

Chapter 11	Page 359
Using a relocatable macroassembler. Using macros and maintaining a library of routines.	

### ---- Section 5    A CP/M Users reference    ----

Appendix A    The CP/M Assemblers. Macros and pseudoops.	Page 374
Appendix B    Introduction to BASIC-E the CP/M Basic.	Page 396
Appendix C    The CP/M Plus implementation on the Amstrad 6128 & 8256	Page 416
Appendix D    The Amstrad Utilities.	Page 464
Appendix E    The internal workings of the CCP - an insight.	Page 475

### ---- Tables of BDOS and BIOS functions    ----

BDOS Character Functions	Page 152
BDOS Drive Functions	Page 166
BDOS FCB and Directory Functions	Page 188
BDOS Date and Time Functions	Page 225
BDOS System Control Block Function	Page 232
BDOS System and Miscellaneous Functions	Page 234
BDOS Pseudo Functions	Page 237
BIOS Character Functions	Page 253



## CHAPTER 1 – Introduction to CP/M

CP/M is a control program and monitor for computers running one or more floppy disks. It is by no means the most sophisticated floppy disk operating system, and one can argue its merits in the academic sense in comparison with its fellows. One fact remains; it is by far the most popular system for 8080, Z80, 8085 chips. Its strengths lie in its robustness, versatility, portability and its flexibility. It has won almost universal acceptance as a general purpose operating system for small machines and has been selected and supported by numerous microcomputer manufacturers and software suppliers, but above all by its users. The state of the art in CP/M is called CP/M Plus, (CP/M+ or CP/M 3). This is the operating system on the Amstrad 6126 and 8256.

CP/M's most important function has been to link standard user procedures and applications to a wide variety of computers.

CP/M is, in fact, a very simple and robust operating system that supports expansion and elaboration with great ease. It is a skilled and calculated compromise between size and versatility: It is very possible to devise a system which stores files rather more economically or which accesses the information more speedily. One can certainly construct an operating system that is more helpful to the user. Whether one can do these things in as little memory space, or in such an adaptable way is another matter.

Several qualities contribute to CP/M's popularity. It reallocates disk space dynamically, uses command files in preference to inbuilt functions, allows program access to its primitive functions and is easily adapted to a different hardware environment. It is not unique in these qualities, only in their combination within a simple and robust system. The fanciest system is useless if it occasionally malfunctions or uses up an excessive proportion of the rather cramped memory addressing space of the eight bit microcomputer.

Although the disc operating system called CP/M is no longer a force to be reckoned with in the newer 16-bit microcomputer industry, it still remains one of the most widely used operating systems and is becoming increasingly used by home computers and hobbyists. The microcomputer market is fiercely competitive and it is perhaps surprising that such an unassuming product as CP/M should have become so ubiquitous.

Gary Kildall wrote CP/M almost by accident. CP/M was the right software in the right place at the right time. It was part of his continuing project to implement PL/M on the 8080 chip. Gary was an experienced consultant who had played a major part in the software development work for the 8080 chip. The operating system itself was initially a by-product of Gary's main work as a compiler writer. At the time, around 1974, Gary was engaged, with some colleagues, in developing software for the new 8080 microprocessor. The PL/M compiler that he was writing as part of his work for Intel needed a simple but effective operating system. CP/M was initially wedded to PL/M, and most of it was written in PL/M.

Much of the work on such projects as the Interp/80, which simulated the 8080 before the chip became a reality, were written in PL/M; the next stage was to be PL/M implemented on the 8080 by cross compilation and the construction of an operating system to run the compiler. It was at about this time that the cheap floppy disk appeared and Gary decided that the operating system should be based on this revolutionary mass storage device. CP/M was born out of the

struggle to make the device work with the 8080.

It was one of the microcomputer industries classic misjudgments when Intel brought the project to a halt. If fate had not intervened, CP/M might have become Intel's operating system. For most of us it was fortunate that the firm was suffering growing pains due to the enormous success of their 8080. No one up to that moment had succeeded in persuading so many transistors onto a small piece of silicon and Intel was, in consequence swamped by the demand for it. Rapid growth meant reorganizing management structures and the decision was taken, incredible in hindsight, to jettison the software projects and disband the brilliant team. CP/M and the PL/M compiler went with Gary, and CP/M gradually developed in his spare time and as part of his teaching work. Gary was based as a lecturer at the Monterey Naval postgraduate school, an institution that was heavily sponsored by Intel at the time. The CP/M story at this time remains rather obscure, but it would seem that the development of the operating system became an exercise for Gary and his postgraduate students. Parts of the original product, even now, are actually in the public domain. CP/M was quite incidental to Gary's lecturing work at the time, and the significance of the product was not realised by anyone. In fact, it was one of the first operating systems that were written specially for the 8 in. floppy disk, and it was one of the few that successfully managed true random access. Because it had been 'kicked about' for some time at the college, it had achieved a measure of reliability that was unusual in products for the 8080 chip at that time.

John Torode, an electronics engineer, became interested in the development of CP/M from its early stages and was responsible for the design of the disk controller. His company, Digital Systems, exhibited the wire-wrapped prototype of the controller board for the Altair computer at a local computer club meeting. To the considerable surprise of the spectators, and the relief of John and Gary, it worked perfectly. In collaboration with John Torode, the first systems using CP/M were marketed, and the reputation of the product spread. Originally, it was John Torode who was the first to market CP/M, with his disk controller boards. John Torode's kits were the first reliable boards to become available for hobbyists and system integrators. By 1976, the reviewers could say that the hardware/software combination had a two year development history.

Gary was not the typical entrepreneur of the eight bit microcomputer industry; the microcomputer world beat a path to his door, and begged him to install the system on their machines. The development of the cheap microcomputer meant that, for the first time, hobbyists could construct their own computers. Suddenly, in the U.S, people who were involved in the computer industry in one way or other could have their own computer at home. The advent of the minifloppy disk meant that computers could be made that could rival the power of commercial minicomputers: but such complex devices are useless without software that can control and monitor the internal interacting operations within the Micro. As always, it proved to be much easier to make the physical hardware of the computer than the controlling program to run it. The floppy disk demanded control at the leading edge of the science of control software. In effect, there were suddenly a great number of 8080 based computers with floppy disk drives looking for adequate software to drive and run them. CP/M fell almost fortuitously into this sudden market vacuum and quickly established a ubiquitous position. Gary was besieged by computer manufacturers wanting CP/M. The labour of installing CP/M convinced Gary that there should be an easier way of wedding operating system to machine. The result was CP/M as we know it, with the separation of BDOS (basic disk operating system) and the BIOS (Basic I/O system). BDOS relied on all its I/O being routed through a jump table at the base of the BIOS. The BIOS was supplied to the hardware manufacturer (or kit builder) in a 'skeletal' form, that showed the source of



all parts of the BIOS module that were common to most hardware, and it was up to the installer to put flesh on the bones to make a working system. The BDOS, on the other hand was never to be altered and was not supplied in source. If the BIOS obeyed the rules and was bug-free, then the BDOS performed properly.

In hindsight, it must seem curious that a dry, efficient product, written for programmers, should have swept the market. In fact, the first people to get their hands on working microcomputers were amateurs whose daytime work was in the computer or electronics industry. The cliché 'friendly' had not been invented, but CP/M did not then hold the confusion that it seems to instill in some of today's more naive users. What was more important then was that the whole thing worked and was very cheap and reliable. If you got into difficulties then, Gary was a telephone call away. Digital Research was a grand name for a garden shed. The operation was run on a shoestring, and when you phoned Digital Research, Gary or Dorothy answered the phone. Many people said at the time that it was the personal touch, and Gary's helpfulness, that made CP/M supreme. It was, of course also because Gary had more experience than anyone on the 8080 chip, due to his early development work for Intel so that CP/M had a lead over its competitors in its evolution.

CP/M 1.4 was the end of a line of development. It was a mature product that enabled the user to run even a 16K disk based computer. Unfortunately, the disks had to be 8 in. running single-sided and single density. Some OEMs tried to patch the BDOS to run the burgeoning minifloppy disks, but the results were unsatisfactory. CP/M 2.2 was Gary's answer to the increasing diversity of mass storage device to reach microcomputers. Effectively, it passed to the BIOS some of the responsibilities formerly assumed by the BDOS and it parameterised, within the BIOS, some of the disk dimensions that had been formerly assumed by the BDOS. In other words, the operating system could be installed on a wider range of mass-storage device.

as a stop-gap product and it really failed to take advantage of the power of the 8086 chip. Its rival, MSDOS (originally called "Quick and Dirty, then 86-DOS") was actually derived from CP/M 1.4 and, like CP/M-86, was a rather hurried product in its original form. Unlike CP/M-86, MSDOS received a great deal of time and resources to convert it into the standard operating system for 8088/6-based micros. Despite the undoubted brand-loyalty that existed amongst users of CP/M, the difference between CP/M86 and MSDOS was too noticeable to ignore. When IBM adopted MSDOS over CP/M for its personal computer, the writing was clearly on the wall for CP/M. Instead of developing CP/M to make it into an operating system to rival MSDOS, Digital Research threw their resources into Concurrent CP/M, a rather more esoteric product that, for all its technical virtues, never caught the imagination of public or even software developers.

Gary's influence in CP/M ceased with CP/M 2.2. CP/M 3, the new 8-bit operating system now called CP/M Plus, was much more the result of teamwork. Although it continued the basic ideas of 2.2, it extended them to enable even more unlikely devices to be used as CP/M mass storage devices. The advent of bank storage enabled the operating system to be greatly expanded, and the spartan CP/M 2.2 was given all the features that were omitted from previous versions because of lack of space. Digital Research adopted many ideas from the rivals to CP/M, such as TurboDos and tried to make CP/M more usable by non-programmers. CP/M Plus was the work of a team, and was initially met with some trepidation by the industry. They need not have worried. Although it needs great skill to install properly, it is a superb product, and should extend the life of eight-bit operating systems for a long time yet.

CP/M-86, the unexciting sixteen-bit product, was due for revision and many people felt that it would be the first product that would receive attention after CP/M-80. Surprisingly, the product that appeared in response to the MSDOS challenge was not a CP/M-86 Plus but Concurrent CP/M. Concurrent CP/M is the result of Digital Research's experience with Multitasking, and evolved from Digital Research's lacklustre operating system MP/M, the multitasking operating system. Concurrent is a version of CP/M that is almost up to the CP/M Plus specification and which allows several tasks to run at the same time. It is as if the user has control over several computers at the same time. Every task continues and the user can see each screen just by pressing the appropriate key. Each task relates to a 'virtual' screen, and the operating system maintains areas of memory for each screen. In a well-implemented system, the screen can maintain a 'window' on several tasks at once and it is possible, in the right hardware environment, to have a screen split into areas, each of which maintains a virtual screen on a task. The development of Concurrent was a rather dismal series of revisions, from 2.0 to 4.1, none of which broke the stranglehold of MSDOS. The windowing was greatly refined to allow the user complete control over the size, colour, and location of each window. Then MSDOS compatibility became the main development goal, with Concurrent v3 able to emulate MSDOS v1 with some degree of success, and Concurrent V4 able to emulate MSDOS V2. Curiously Concurrent still did not manage all the features of CP/M Plus, such as the editing of the command line. CP/M-86 Plus eventually emerged in 1985, called DOS Plus, two years later than expected, with added MSDOS emulation. It had the facility to manage background tasks and promised a simpler, and less esoteric product than Concurrent. It is equivalent to CP/M Plus and has all the latter's advantages, but it also manages up to four background tasks. Unlike Concurrent, virtual screens are not maintained for the other tasks, so the type of background task will be limited to activities that are not dependent on a console, such as print spooling, compilation, or telecommunication. Meanwhile Concurrent too had suffered a number of name changes, from Concurrent CP/M to Concurrent PCDOS, via Concurrent MP/M and Concurrent DOS. These name changes seemed to do nothing to enhance the popularity of the product.

Networking is an aspect of CP/M that has been less than successful for Digital Research. The combination of MP/M v1 and CP/NET was one that still makes OEMs flinch. It was possible to get it to work, but it was never satisfactory. MP/M v1 was a rather horrid product, not because it was particularly unsound, but because it was hailed as a way of providing Multiuser facilities on an 8 bit micro. It was originally seen as a multitasking system, and as such, it was not that bad, but it gained a justified tarnish in general use as a multiuser system. MP/M-II and MP/M-86 emerged to put things right. These were good products, and are becoming increasingly popular, but the resources to make MP/M and CP/NET into a workable network were not there. The skills needed to install a MP/M network are beyond the internal programming resources of most OEMs. When done properly, the network was very effective, but there are constraints. Concurrent CP/M was the obvious vehicle for a network and a great deal of work has gone into developing DRNet into an effective product. Concurrent 4.1 now has the networking 'built-in' but, as I write, it is too early to say whether it will prove itself.

By the beginning of 1982, it became apparent that there would be no single chip that would dominate the 16 bit market. For a company whose lifeblood was operating systems, with all the associated assemblers, linkers, and debuggers, this was a worrying time. Digital Research's ties with Intel assured that the 8086 series would have all the initial attention, but the Motorola 68000, Natsemi/Fairchild 16000, and Zilog Z8000 all required CP/M. To the surprise of the pundits, Unix failed to attract the average user, despite its intrinsic

portability. The punters, at that time, wanted CP/M on their new sixteen-bit computers.

Writing CP/M for each chip would have been a gigantic task. The decision was taken to borrow UNIX's great strength, the programming language C. CP/M was rewritten in C. All that was then needed for each chip would then be an efficient C compiler. Portable CP/M exists. It is relatively slow and bulky. Whether it is the basis for a successful product range is still not clear. CP/M-68K, developed for the Motorola 6800 series took a long time to take off, though it has now been adopted by Atari for their new range. CP/M-16K, for the NatSemi 16000 series and the CP/M for the Z8000 never seemed to appear at all, and CP/M remains dominant only in the eight-bit environment.

Currently, the work at Digital Research focuses on applications as well as the new operating system Concurrent 286. GEM provides the basis to these applications packages that are obvious imitations of the software environment developed for Apple's Macintosh. These are, in turn, based on the Smalltalk operating system from Xerox. GEM enables a micro to use graphic icon and pull-down menus, and gives application programs a number of useful graphics facilities. GEM evolved from Digital Research's work with the GKS graphics interface standard. Originally, Digital Research released a bought-in product called GSX. (GEM is actually GSX v2). This provided an interface that provided the basis for a standard graphics interface. It quickly became apparent that it was difficult for programmers to come to grips with, and those that managed it found that it did not support the features that they wanted and was slow in performance. One hopes that the years of work at Digital Research devoted to a graphics-based DOS will bear fruit in the GEM range. Success will depend on whether third-party software developers adopt the GEM interface and facilities, in order to produce GEM-based programs. Digital Research are releasing a range of GEM products including GEM Draw, GEM Paint and GEM Graph to start the ball rolling.



## CHAPTER 2 - CP/M software

It is not due to any spectacular virtues as an operating system that CP/M has been so popular. It is not the most attractive or easiest operating to use; it borrows extensively from the old minicomputer operating systems in its interface and never had startlingly original ideas. What, then, makes it so attractive? It is obligatory as an eight-bit operating system because of the applications running under it. To get at the applications (such as spreadsheeting, wordprocessing, database, and accounting) you need CP/M.

What CP/M does is to turn a computer into a 'standard' computer that will run the bulk of published software. It provides a standard interface between the hardware and the software, irrespective of who made the computer. A CP/M computer is any computer capable of running CP/M in any of its guises. A program written for CP/M-80 will be likely to run on any computer that has a version of CP/M-80 (ie CP/M 1.4, CP/M 2.2 or CP/M Plus) but will not run on any version of CP/M-86 (or Concurrent) nor would it run under CP/M-68K. This is because the program is designed for a particular CPU and the operating system can do little to protect the program from that (unlike the UCSD P system). All the operating system does is to establish a standard interface with the I/O (console, ports, discs etc).

CP/M-80 is the name by which the eight-bit CP/M products are known. Different versions of CP/M (eg CP/M-80, CP/M-86 or CP/M-68K) present the user with a slightly different interface, and different revisions of CP/M (eg CP/M 1.4, CP/M 2.2 or CP/M Plus) contain more, or fewer of the features but have 'backward compatibility' in that programs written for the older CP/M 1.4 will work under CP/M Plus).

Until recently, all microcomputers had 8-bit microprocessors. Most large 'business' microcomputers used CP/M to provide a standard interface and enable them to run the programs that were being marketed. Because CP/M arrived so early in the development of the microcomputer it gained a dominant position in the industry. Software developers made sure that their programs would work on any computer that ran CP/M, because there were more CP/M based computers than any others out there: Manufacturers chose CP/M when developing new computers because there were so many CP/M programs available. In the wake of the popularity of CP/M, many imitations of CP/M appeared. Most of them never gained popularity beyond the original hardware that they appeared on. CDOS, for example, was tailored on Cromemco hardware. CDOS was not important in itself, but Cromemco produced several excellent software products to run under CDOS. It was, however, possible to run them in CP/M with a conversion utility similar to an RSX to convert the BDOS calls. As CP/M software is more easily obtainable now, CDOS, which was derived from CP/M 1.3 has faded into obscurity. Cromemco are now wedded to a Unix operating system. TPM was another CP/M lookalike, significant only in the days before CP/M Plus, when TPMs advanced features made it attractive. Turbodos is another matter. It was a great advance because of its multitasking structure and its accent on networking. It has continued to develop and has a 16-bit version with PCDOS emulation. In many areas it has features superior to CP/M. MSDOS (and PCDOS) started out as CP/M lookalikes for 16-bit computers. Its interface is very similar, but it has now got Unix-like extensions that make interfacing rather simpler than CP/M. It has a less efficient means of disc allocation, but a better bdos interface. Its chief superiority is the fact that it was chosen by IBM as the operating system for their PCs.

There were certain programs, running with CP/M, that became extremely popular. Obviously, an operating system is only as good as the software running under it and the popularity of these programs cemented the grip of CP/M on the market. The best operating systems do not intrude themselves on the user and merely support applications such as wordprocessing, games or bookkeeping as efficiently as possible. CP/M's great strength is the ease with which it can be used by such applications. From the earliest times of the CP/M scene, it was taken up by a wide spectrum of users. whereas UNIX gained its power base only in the universities, and Pick only in commercial use, CP/M was enthusiastically adopted by hobbyists, businesses, universities and research. The result was a rich variety of programs developed to run under CP/M, not only commercially marketed for business but also contributed to the public domain by universities, hobbyists, and research departments. Today, there exists a huge amount of free programs for CP/M-80, and more are becoming available every day. Although CP/M is no longer the feted system for business machines, its future is assured because of its ubiquity and the huge and heterogeneous user base.

Certain programs became part of the CP/M scene and, as such, deserve description as much as the operating system itself.

**WORDSTAR** is the standard wordprocessor running under CP/M. Micropro's Wordstar was software of the highest quality. Although there have been other comparable wordprocessor programs since, it reigned supreme for a long time. Wordstar was the first popular program that used the terminal or computer screen to the fullest extent, but allowed the user to 'configure' or alter the screen to allow the program to run with different terminals. CP/M was written before the advent of cheap 'full-feature' terminals and, whilst it managed the standard interface into the disk hardware, it failed to provide the standard interface into particular character or graphics screens, all of which had their own conventions for moving the cursor, blanking the screen, erasing lines etc. Before WordStar, obtaining wordprocessing facilities required the purchase of expensive wordprocessing machinery. The disadvantage of the dedicated wordprocessor was that it could not be used for any other purpose and the files created by the wordprocessor could not be accessed by any other application. Wordstar turned any CP/M computer into a wordprocessor. On comparative tests, Wordstar stood up to the specialized rivals, and became deservedly popular. The computer it was used on could be used for other purposes, and the cost of installation was less than was the case for the dedicated wordprocessor.

**MBASIC** from Microsoft became the standard BASIC for micros. It was Microsoft's first product and the first full BASIC to appear on a Micro. It ran under CP/M. It kept ahead of its competitors because Microsoft was able to release a matching compiler. BASIC is an interpretive language and MBASIC was, of course, an interpreter. However, it was difficult to write commercial programs for an interpretive language, as one had to give away the source code along with the program and thereby risk competitors incorporating ideas gleaned from it. Survival in such a competitive industry is difficult enough without ones rivals standing on ones shoulders. MBASIC had a matching compiler called BASCOM which was reasonably compatible and, instead of interpreting the source, it compiled it into an executable '.COM' file. Obviously, one could not reconstitute the source file from the '.COM' file, and there was an increase in speed of execution of the program. The user also did not need to purchase the interpreter to run the program, either. BASCOM turns BASIC into an effective language that rivals the more traditional compilers such as FORTRAN in its performance. It is particularly attractive to be able to develop a language in a cozy interpretive environment and then compile the results to get the best performance. This arrangement has recently been offered by 16-bit C interpreters, but the 8-bit micros are not used to such luxuries.

BASIC-E was originally distributed free with CP/M 1.4. It was a good robust BASIC whose main advantage was its ability to run in the restricted memory of the early micros. It was classed as being 'a semi-compiler'. This was misleading as, in reality, it only compiled into an intermediate code that then had to be interpreted. (similar to the UCSD 'P' system). This method enabled it to run longer programs or run in a small TPA (transient Program Area). As the documentation for BASIC-E has not had such a widespread distribution as the language itself, we will devote a whole chapter to it in this book. BASIC-E was a bit limiting, particularly in its file handling and was superseded by CBASIC, which was not in the public domain. The general design was the same, but the author added features and improved the file-handling. Rather later on, a true compiler was added to CBASIC, called CB-80. This conferred on CBASIC/BASIC-E/CB-80 some of the advantages of MBASIC, though a truly interpretive CBASIC never appeared. As the former had structured features (including eliminating the need for line numbers) it became a rather superior product to Microsoft's offering. CB-80 has evolved into a language that is still compatible with BASIC but has such greater flexibility that it is almost a new language. You can redimension arrays as often as you like, access random-access files sequentially, use labels instead of line numbers, use local variables in multi-line functions, and access functions contained in other modules. In fact, through half closed eyes, it is beginning to look like Pascal. There is a 16-bit version, so your programs are relatively future-proof.

DBASE-II was the first program that allowed the end-user to write applications that involved sophisticated file-handling. BASIC offered simple file handling and was not easy for a non-programmer to use. There were attempts to remedy this with such products as KBASIC, which had built-in KSAM (Keyed Sequential Access Method) facilities. It occurred to several people that a new language was required that understood file-handling, screen addressing, data input, and so on, but was easy enough for the end-user to operate. This meant that the end user could construct such things as stock control programs, 'electronic card indexes' or so on, to his taste, and alter, update and improve the program as required. There have been many applications languages before, and they are rapidly ousting traditional computer languages in popularity on Mini and mainframes. What made DBASE-II significant was the DIY approach. This had great appeal to the microcomputer user who had previously had perforce to employ a programmer to do this sort of work for him. DBASE-II was the first of the interpretive database programs and was followed by such others as 'Sensible Solution' and 'KnowledgeMan'. DBASE-II attracted other software houses to write support products, including a compiler, and code generator. The effect was revolutionary, and made real computing power available to more and more users at a much lower cost. DBASE-II was an immensely liberating product.

VISICALC was the first 'spreadsheet' program. It is strange that the computer industry was, until recently, completely innocent of such a fundamental tool as the spreadsheet, but it required considerable crossfertilization between computer science and the Real World. A systems-analyst went on a course in Management skills, and, in the course of a gruelling and tedious lecture in simple accounting, realized that it must be simpler to do on a microcomputer. The whole idea of dividing up paper into columns and rows, and processing the intersections so dear to accountants, gave birth to the 'spreadsheet' computer model. Although it is a rather foreign idea for computer science, it is easy for the end-user to grasp. The systems-analyst teamed up with a particularly gifted programmer and Visicalc was born. Originally it was written for a cassette-tape Apple II and single-handedly made Apple Corp's fortune. Everyone wanted Visicalc, and so bought Apple IIs. For a



long time, it was unavailable on CP/M, and so imitations became available before the real thing. By the time Visicalc had become available, the imitations such as SuperCalc and Multiplan had become entrenched and Visicalc made very little impact. The imitations were, as is often the case, superior to the original, and so Visicalc is no longer available. Multiplan is probably the best, and reckoned to be virtually indispensable for the business user. Supercalc is preferred by many as it is rather simpler and cheaper. It also has a big brother in the 16-bit Supercalc-3 that has the same user interface.

BSTAM was the first standard communications program for CP/M. Where two Users of CP/M computers wish to exchange files reliably via a telephone line they must use programs that share the same protocol. Until comparatively recently, there was only one popular file transfer utility, and that was BSTAM. Using it is simplicity itself. One makes the physical connection, and types TRANSMIT <filename> at one end and RECEIVE <filename> at the other. the actual protocol used by BSTAM has never been made public so that it is only possible to use BSTAM if both ends have it. XMODEM, MODEM7, or UKM7 is superior in every way to BSTAM and is in the public domain. It is slightly more difficult to use but its protocol is public knowledge. Most modern communications programs offer the choice of this protocol, and so BSTAM is now going out of use. Other utilities such as ASCOM or MOVE-IT introduced their own protocols but they have not achieved the same popularity as XMODEM.

GSS-GRAPH, or DR-GRAPH as it is now called, was the first good portable business-graphics program to run under CP/M. It uses GSX as its interface into the actual graphics hardware and is therefore capable of writing graphs to screen, printer or plotter with equal facility. GSS-GRAPH is the only significant available program to run under GSX. It runs slowly and is limited in the type of data it will input, but it is nevertheless a workmanlike program written by knowledgeable programmers. For an 8-bit computer, it is still unrivalled as a business graphics program for producing pie-charts, scatter graphs, histograms or line graphs. To understand the enormous value of GSS-GRAPH, one has to have tried to do them by hand. One of the authors of this book had to undertake a research project that was published as a book, lavishly illustrated by graphs. The speed of the production of graphs under GSS-GRAPH may be slow, but it is insignificant compared with the day it took to get each hand-produced graph to publishable quality! There may be better graphics programs around on 16-bit microcomputers, utilizing the greater available memory, but DR-GRAPH(GSS-GRAPH ) is a good, workmanlike product for people who are serious about producing graphs.

Towards the end of the first half of the decade, the spreadsheet, database, wordprocessor and communications utility became the most commonly purchased type of program to run on a micro. All of these programs manipulated data, and it was becoming increasingly apparent that the data produced by one was usually indigestible by another. GSS-GRAPH could not read Multiplan data, for example, and Wordstar could not access a mailing list produced by DBASE-II. It was impossible to produce a program to combine all the functions of these, and thereby ensure data compatibility, on an 8-bit micro: there was just not enough available memory. However, it was certainly possible to develop an 'integrated' package for 16-bit computers because they could address much more memory and both programs and data could therefore be bigger. The advent of the new integrated packages such as 'LOTUS 123' spelt the end of the use of 8-bit computers in the business environment. They made much more sense, and required much less effort to learn. One could send the results of the spreadsheet to the database, or vice-versa, one could put the result of a spreadsheet calculation or a database search into a report and edit it with the wordprocessor, or one could send the results to the graphics program to produce a graph. Such

flexibility is not possible within existing 8-bit software technology. Address space is just too cramped. It might be possible as overlay techniques and compilers improve, but then, 16-bit computers are becoming cheaper so why bother? It is for this reason, not for the superiority of 16-bit operating systems or 8088-based micros, that the commercial world has abandoned 8-bit computers and CP/M.

Z80 micros are still capable of the work they managed during the height of their use in the business environment, before the advent of PCDOS. There are fewer new business-oriented programs being produced for Z80 computers but the 'greats' such as Wordstar, DBASE-II and Multiplan are still with us. Even now, the full potential of the Z80 has yet to be realized. The construction of a language compiler as efficient as Turbo Pascal, for example, would have seemed impossible only a short while ago. The construction of Z80 games software is becoming so sophisticated as to appear almost magical. There is life in the old chip yet!

## CHAPTER 3 – Beginner's guide to CP/M

If you are a pundit, well versed in the mysteries of such esoteric commands as:-

```
PIP AUX:=M:DIKG.C(T10UZN80)
```

then this chapter is not for you: go instead to the chapter on the BDOS or RSXS for your amusement.

This chapter is intended for those people who have unwrapped the computer for the first time and are wondering, with slight rising feelings of panic, how to operate it. We have all faced this moment. People will tell you that CP/M, the program that makes everything else work in your micro, is complicated. This isn't really so if you are doing simple things. You will be able to explore its more advanced features after a while. To get started is more simple. When you have been fortified by this chapter, it is safe to venture into chapter four, where we slightly loosen the firm grip by which we hold your hand. We must emphasize strongly that the only way to learn CP/M is by using it, so if you are reading this in a nice comfy armchair, with a hot-water bottle in your lap and your pet dog at your foot, resting its chin on your shoe, gazing devotedly up at you....GET UP...go to the computer and try out some of the things we describe in the next two chapters. If, on the other hand, you have bought a computer to run a payroll system; it is friday, and a hoard of restless navvies are pacing up and down outside the room waiting for their pay. You are hiding behind a filing cabinet, desperately thumbing through this book to find out how to work the computer; then you should immediately turn to the 'how to..' section of this chapter.

### 3.1 What a Computer is - and does.

The word 'computer' arouses such a mixture of emotions. In the past few years, computers have intruded themselves into every corner of our lives; in the homes, in industry and commerce. They have removed some of the most dreadfully tedious clerical tasks, and removed the need for much boring repetitive industrial work. They are responsible for removing so much human drudgery that one might think they would be thought of as a universal blessing. But their image has been mixed: thankfulness for their labour-saving powers has been mingled with fear of redundancy, and wonder at their sheer efficiency has never been free from suspicion that we might have invited a cuckoo into the nest, one that will outsmart its inventors - a suspicion mightily exploited by the science fiction writers. Computers may have liberated us from the drudgery of clerical operations, but have taken much of the humanity away from them.

So let there be stated a sentiment you must have heard a dozen times, that computers are basically dumb. They can only do three functions: add, subtract and compare. The fact that they can do these things at the speed of light doesn't alter the fact that they couldn't tell Ghandi from Crossroads, and if the Amstrad microcomputer which you have shown such excellent discernment in buying seems incredibly clever, it is only the humans who have programmed it who have made it so. Any mental wrestling this computer may involve you in will be a tussle with the programmers, not the machine. Always the machine is subordinate to the person, and that thought is carried through to the way this manual is written. Information is served up at the level and in the order in which it is felt you want it, not in such a way that it shows you how clever it

is - and leaves you baffled. So, in keeping with that thought, there will at first be plenty of analogies from everyday life, none of them too desperately serious, so that you can get the hang of the mysterious ways of a computer by comparing it with concepts that are already familiar.

So what is CP/M? It is a microcomputer operating system. Every computer has an Operating System - and so do you. If you had charge of the mailing of a monthly magazine, you wouldn't rush at it, scribbling addresses and reminders as they came into your head. You would soon devise a procedure for recording addresses, incorporating new subscribers, affixing labels to envelopes and bundling them up according to postcodes, inserting reminder slips for those whose subscriptions are due. It would almost certainly differ from another person's system; neither may be perfect; there may not even be a perfect operating system; but it is a system and it works. Your Amstrad microcomputer works on an operating system known as CP/M. No, you won't find out yet what it stands for because there's no need to know. We did promise! What is useful to know is the concept of CP/M, which has done for computers roughly what McDonalds have done for eating out. If you arrive in a strange town in the evening and need a meal, you know that a McDonalds establishment will provide you with a predictable standard of cooking, service, decor and price. They have standardized their product so effectively that they do not appear to be subject to the same pressures as other restaurants. In fact, of course, they buy a lot of their food locally, sometimes paying less, sometimes more, and have to compete for staff like everyone else. But by presenting the same uniform image nationwide, one could say that they have 'masked' local conditions. CP/M has similarly 'masked' the various makes of computer (hardware) that it runs on - for there are many - so that any CP/M operator, such as you will be within 20 minutes, will be perfectly at home on a strange machine, because the commands that are punched in will be virtually identical.

There is another benefit there. Even the best computer, supported by the best manual, will not tell you everything at once. Nor could you absorb it if they did. There is most definitely a 'learning curve' with computers. For that reason it is very much worth keeping in regular touch with other CP/M users, who are plentiful, so that they can feed the growing edge of your knowledge. Though you could scarcely credit it, you will be pleasantly surprised how soon it will be before you are passing on the odd nugget!

This perhaps gives an idea of what CP/M does for computers (also why it is the most popular system) but not what it does for you. Think of a DIY workshop. On one hand there are the raw materials - planks, hardboard, glass. On the other there are the tools. The CP/M programs and the Amstrad utilities are the toolkit. The raw materials they operate on are sometimes plain words, knocking them into letter format, complete with tabs, neatly paged lengths and justified margins; sometimes they are numbers, which are rattled out by the program in the form of an immaculate balance sheet, with all the columns and rows adding up and balancing automatically.

Those two last examples are the sort of applications for a computer that are so common that special Applications Programs have been written to cope with them. They sound complicated, and they are, but all the complexities have been kept firmly under the lid. For the actual user they are simplicity itself. Punch in the month's figures, press a button or two and out rolls the balance sheet in whatever form you have chosen. They may only be able to do one trick, but they do it very conveniently. To use a jargon word you will meet, the complexity of the program has been masked, to make it 'user-friendly'. Or, to continue the 'tools and materials' analogy, the computer is supplied with a beginner's toolkit, equivalent to a hammer, screwdriver etc, for general use



(these are the 'utilities'). Applications programs are the equivalent of the planer and hedgetrimmer attachment - single purpose dedicated tools bought in later.

To return to the opening sentiments, computer manuals have come a long way since the sparse, impenetrable and often illiterate documentation of the early days, which may have sufficed for the enthusiastic hobbyist and the professional programmer (though the consensus is that even they were far from happy), but it left a huge gap in the middle. This gap comprises the workaday commercial world, abounding in resourceful, intelligent businessmen ever seeking a new market advantage - a quest in which becoming computerized is mostly taken for granted - yet where commercial pressures preclude all but a few becoming fluent in computer programming. Nor should one forget the equally intelligent typists who mainly have to operate these strange new devices, who likewise have no wish ever to write a program, yet who have enough British inventiveness not to be content with unnecessary, expensive programs 'specially written' for them by dubious software houses, when they rightly suspect that a bit of ingenuity and explorer's courage could have produced the same results from what they already have. Though the professional programmer will be more than happy with the power of the Amstrad 6128 or PCW 8256, it will be equally at home in the modern office or laboratory, by virtue of its versatility and speed, the built-in utilities and - not least - a comprehensive manual.

You may well find that the specialist easy-to-use application programs mentioned above, like Wordstar, Multiplan and dBase II, provide your daily diet, so much so that you are tempted never to descend into the strange ways and phrases of CP/M. It is understandable at first to stick to one or two easy and stereotyped procedures, but the CP/M toolkit is a much more efficient way of coping with what one might call household chores: copying files from one disk to another, lumping strings of commands together, rather than doing them one by one with the specialist programs, which always run more slowly anyway. Don't be afraid to experiment - your Amstrad micro will give you a lot of fun, as well as a great deal of value.

### 3.2 Jargon - or shorthand?

To the novice, special 'buzz words' encountered in any new subject inevitably seem bewildering and at times deliberately baffling. Computer language must be one of the chief culprits. There is no denying that much of the language, though colourful, is at times both illogical and counterproductive. It positively puts people off finding out more about the subject. In every discipline, though, there is always a case for sensible shorthand. It is that, and that only, which we would like to introduce now.

### 3.3 Abstractions

Sooner or later the maths teacher has to give up saying 'suppose there are 3 oranges or 4 bananas' in favour of 'let the number of oranges be  $x$  and let the number of bananas be  $y$ '; in other words he has to introduce abstract concepts. In the same way, for brevity's sake, instead of giving easy to understand names to a file like Jones and Smith, we must start referring to a file name simply as `fn`.

### 3.4 Filenames

In fact there are two broad groups of file names, ambiguous and unambiguous, which we shall refer to as afn and ufn. Smith2.ltr is an unambiguous file name because, as its name implies, it is unique. It represents the second letter in the correspondence with Smith. In fact there can only be one file of that name, because if you tried to create another it would be rejected.

### 3.5 Useful ambiguity

For handling purposes, though, it is often useful to be able to deal with a whole batch of files of a similar type. For this the wildcard symbols \* and ? are used. For example if after the first few weeks of heavy computing, when everything goes onto the same disc, you decide you want to tidy up a bit, it may seem sensible to transfer all the letter files onto a separate disc. Instead of doing this file by file, typing out the same transfer command over and over again, all that needs to be done is transfer the general type of file \*.ltr. The symbol \* in this case means anything before the dot, no matter how long or short the label is.

Another example: If you have a disk for customer inquiries, and you notice that Jones has now become a regular customer and must be given a more permanent niche, you would transfer the general type file Jones\*.\*. Perhaps you've lost him. In which case erase Jones\*.\*. The symbol ?, on the other hand, stands for a particular character. If you have electrical and mechanical goods at stores in Bath, York and Canterbury, and your dealings with these three stores involve PARTs list, INvoices and LetTeRs, your files might look like this: EBATH659.INV, MYORK51.LTR, MCANT12.PAR, and so forth. All invoices to York would be covered by ?York\*.inv; all letters to all branches on electrical matters would be dealt with by E\*.LTR; dealings with Canterbury on any topic at all would be covered by ?CANT\*.\*

The reason why these symbols are nicknamed 'wild cards' should be fairly obvious. Certain operations, such as naming, renaming or copying a file, naturally only work with specific, unambiguous files. Common sense dictates really when 'blanket' commands are possible.

### 3.6 Control C

Starting the computer from cold, it is easy enough to see what happens: the system has a kind of self starter mechanism built into the first track, which automatically loads CP/M into memory. When working a computer, especially at first, it is easy to get into a muddle and it is then very tempting to remove the disks, switch the machine off and start again. This is not always necessary. When a jet aircraft 'flames out' at high altitude, it does not necessarily crash. It does not even have to land. The pilot still has plenty going for him: lots of height and most of his systems are still working; all he needs to do is merely to relight his engines and proceed on his way ('merely!'). The equivalent in computers is to press (simultaneously) CONTROL and the character C. This is abbreviated to ^C. In most cases this will get you out of whatever muddle you are in, and get you back again to the PROMPT sign. You may lose what you had just been working on, but everything else should still be intact. If you switch off and then on again, however, you lose everything that was held in temporary memory. Pressing ^C is known as a 'warm boot', whereas the 'start up from cold' procedure is known, naturally, as a

'cold boot'.

### 3.7 The Ejector Seat

There are occasions when a wrong sequence or wrong keying can cause the computer to 'lock up'. In this state, neither ^C nor any other combination of keys has any effect. If you press down the three keys Control, shift and ESC then this will force the micro back to BASIC. Insert the System Disk in the drive (A), if it is not already there; then type |CPM. This will restart CP/M. You will probably lose the file you were just working on, and you would have to reconfigure temporary settings of e.g. keys, I/O port settings and the like.

### 3.8 Syntax

Computers can be as maddening as Manuel in Fawlty Towers. You pound out a really impressive stream of commands, press carriage return, and up comes simply a '?'. You can almost hear it saying Che? The reason invariably is that you have got something slightly wrong and that is good enough for the computer to make it look the other way. If you meant:

```
type Smith.ltr
```

then

```
type Smitth.ltr
```

simply will not do. Nor will it do if the order is wrong. If you meant:

```
pip a:travel.doc=b:travel.doc
```

and you write

```
pip b:travel.doc=a:travel.doc
```

it will be rejected. Even if you are a careful typist, your problems are not solved because many of the commands are long and complex, so if the computer throws them back at you, it is well worth while consulting the User's Guide to check if you are using the correct Syntax, especially spaces and commas, and lack of spaces and commas.

One general point about correct order may be made here: when heavy packages are being passed between people, you often hear 'to you, from me' called out in a sing-song voice, for safety's sake. This is always the system in which computers work: first the destination, then the source. The more usual phrase 'from A to B' is the order that is never used. CP/M works on the idea of 'assignment'. B:=A: really means LET B:=A:.

### 3.9 Toggle

This is the principle of the bathroom light switch. Switching the light on, and off again, is done by the same operation of pulling on the cord. Most ballpoint pens operate similarly. ^P therefore means 'start printing'; when next so used, it means 'cease print'. This is a toggle.

### 3.11 String

Usually a short piece of text, perhaps only a single character, but the point is that it is text and not part of a command. For example:

```
pip blenheim.xtr=blenheim.doc[Next year^Zdecade.^Z]<cr>
```

is a complex command to extract part of a main document and make it into a

separate file. The meaning of the command is of no interest here, but it illustrates how the computer is being told to look out for 'strings' to mark the start and finish points of the required extract - in this case the strings are the phrases 'Next year' and 'decade.'.

### 3.12 Default

There are so many options available for computer procedures that rather than specify each and every one, the most commonly used are pre-set to what are called 'default' conditions. For example, in Wordstar the line measure default is 65 spaces and has to be altered if you wish to make it different. In everyday life, if you just ask for 'a rail ticket' the default type you are given is second class. This use of the word default here has nothing to do with winning tennis matches because your opponent doesn't turn up.

#### Yes/No?

In most computer operations you have to type in precisely what you want it to do for you. With some processes, such as applications programs, the work has been done for you and you are presented with simple yes or no choices - often abbreviated:

#### ERASE B:.\* (Y/N)?

Y/N? is short for "Yes or No?". Sometimes simply pressing the y or n key is enough. Otherwise a <cr> as well is needed. (the return key is often described to children as the 'please or thankyou button'.

### 3.13 Software and Hardware

The old maxim: 'if you can kick it, it's hardware' is difficult to better. The computer, disk drives and printer are all hardware. The programs that run on the system are the software. If you like, in a hi-fi system, the turntable, amplifier and speakers are the hardware; the Beethoven 9th that it is being played is the software. (The floppy disks and records are the 'media'.) One further point in particular: since CP/M is the most popular system, and 8 byte machines the most common, there is available to you the widest range of cheap software on the market.

### 3.14 Error Messages

From time to time you will type in a wrong command that the computer cannot obey. If it is a plain typing error, a plain '?' will appear, inviting you to try again. Perhaps you have called correctly for a file, but which is not present on the disk drive you have specified. In this case a printed message will appear, pointing out the error of your ways. This is especially so with the applications and utility programs, which are characterized by the work done on them behind the scenes to make them easy to use. We have come a long way from the early days of CP/M, hinted at in the dark remark in the second introduction, when almost any minor felony brought up 'BDOS error on B', leaving the operator to wonder which of scores of errors that could refer to. Unfortunately CP/M acquired something of a bad reputation for this, but it is no longer justified.



### 3.15 Synonyms

Most other explanations can wait until they become necessary. It is sufficient now to explain one possible point of confusion about names of things. In the opening chapters we try to explain matters in terms that would be understood by someone who only knows only typewriters and other traditional pieces of equipment. As the manual progresses, and you develop from a typist into a computer expert, the language understandably takes on more of a computing flavour. For example, we have used the phrase 'the screen', because of familiarity with TV and because it is the most easily understood word. But it is often (more correctly) referred to in manuals as the CRT (Cathode Ray Tube) or the Console. The non expert punches keys and sees things appear on the screen, and assumes that this is where it is all happening. In fact it is more like a modern signal box, where the railwayman flicks a switch and lights flash to indicate that points have been changed, but the action is really taking place a mile or two down the track. He is only operating a console. Explanations of how and why things happen will only be found in other chapters.

### 3.16 The keyboard and screen

It looks like a typewriter keyboard, but is it? All those extra bits round the edge make it seem a good deal more sinister. But essentially the 6128 keyboard is designed for typists and nine tenths of the keys you will strike are familiar 'qwerty' keys in the normal place.

Alphabet. a-z and (in shift) A-Z are exactly as you would expect. (See Caps Lock, though)

Numerals. The convention of numeral keys along the top row is duplicated by another set on the right that are also preceded by an f to show that they can also be used as 'Function Keys'. Either set can be used, but in practice touch typing is easier with the calculator style layout, which can make a deal of difference when entering lists of numbers.

Shift works as on normal typewriter.

Tab. The tab key moves the cursor to the next tab position. Tabs usually occur every 8 spaces.

The Caps Lock key, when pressed once, converts all lower case characters into upper case. To reset this, press the key again. In other words, a-z are converted to A-Z; but numerals and symbols, such as '[' are not. 5 remains 5 and does not become %. The shift key will work as usual on the non-alphabetic keys. To engage Caps Lock, press once. Pressing again disengages.

CR. Carriage Return works as on a normal typewriter, but it has another most important function. If a command is typed in, which is going to operate on some data, the CR is the key that initiates the action. It is the 'now do it' button. It is denoted by the symbol <cr>. There is another seemingly hairsplitting point to make: in typewriter operations, whether it is a golf ball fitment or a conventional roller, 'carriage return' also means rolling up a line - 'line feed'. In computer use these are often treated separately. Strictly, a carriage return only needs to mean that the typing point moves from right to left, though in fact the <cr> key carries out a line feed as well.

### 3.17 The Control Key

The Control Key works similarly to the principle of the Shift key familiar to typists, in that pressing a key in Shift has a completely different effect to pressing it in unshift. Same with Control, but the effect is mostly to produce commands rather than characters.

If it seems that in computer work we now have to remember Three sets of keyboard characters in place of the usual two, rest assured it is not nearly as complex as that.

The effects of pressing Control and a character key are discussed more fully later. Both keys have to be depressed simultaneously as with Shift. There is no equivalent to Shift Lock, though, and indeed there are only a few characters that can be linked with Control in this way. The conventional abbreviation for 'Control and character C', for example, is ^C, but this is never achieved by entering the ^ symbol, followed by the character C.

Cursor Controls (Up, down, right and left arrows). The cursor is usually a winking square on the screen, which shows where the next keystroke would be placed. We are now beginning to depart from the manual typewriter layout. Instead of only the paper moving about, while the keys strike always in the same place, the keys now 'strike' wherever the cursor happens to be. The cursor control keys move the cursor in the obvious directions shown by the arrows.

### 3.18 Words on floppy disks

Disks are normally sold in boxes of ten. A 'naked' floppy disk is in fact extremely floppy, and its protective envelope is as much for stiffening (and cleaning) purposes as for obvious protection. On the Amstrad computers, each single side holds about 175,000 characters.

Regarding care of disks themselves, they will be handled all day every day, and obviously they have been made to stand reasonable wear and tear. Some precautions must never be laid aside, however:

- 1/ They must never have hot coffee mugs stood on them. Exposure to direct sunlight or a hot radiator can do equal damage.
- 2/ Anything magnetic near a floppy is death to whatever is stored on it.
- 3/ Dust and cigarette ash are almost as bad.
- 4/ Telephones are worse, especially the modern cordless ones. The ringing action corrupts any disk lying underneath immediately.

#### 'Housekeeping'

Labels should be affixed in the space provided, and if possible should be written on before affixing. Labelling should be a painstaking and continuous operation. Everyone mocks those offices that have filing drawers marked 'Miscellaneous', but there is many a high powered computerized office whose disk recording system - or lack of it - is equally haphazard.

Always assume the worst - make back-up copies of disks ALL THE TIME.

### 3.19 Powering Up

Take the copy of your System Disk out of its protective box. Insert it into the drive, holding it by the label and with the label side uppermost. Push it firmly home with your thumb. After it has fully disappeared into the slot, the drive will appear to give a final gulp and hold the disk firmly between its teeth. In other words, there will be a click and the disk will be held firmly by the mechanism.

First switch on. The switch is on the back of the keyboard, next to the volume control.

When the disk is home, you must type the line

```
|CPM<cr>
```

if you have a 6128, and the drive will take a look at whatever has been inserted, and because this particular disk has a self-loading routine stored on the first track, it will proceed to help itself. After half a minute, the screen will show:

```
CP/M Plus  Amstrad Consumer Electronics Plc
```

```
v 1.0 61K TPA, 2 disk drives, 1 serial port
```

```
A>
```

to indicate that the computer is now 'up and running'. This start-up procedure is often called 'putting its brains in'. Not a bad description. The machine by itself can only run BASIC. Only when the CP/M operating system software has been loaded on board does it become a (very powerful) general-purpose microcomputer.

### Shutting down

There are two essentials:

- 1/ Never switch off with a disk still in a drive. Push the drive buttons, remove the disks and only then switch off at the side.
- 2/ Always check the RAM drive if you have one, to see if there is anything there that needs preserving. It will otherwise be lost on shutdown. See 'How to copy a file' (to a disk in drives A or B) in Section 3.23.9.

At the risk of sounding preachy, good habits are usually acquired at the beginning of an enterprise or not at all. Why not make a list of procedures, and stick to them? The most important is a good filing system. In the middle of a mammoth task, files tend to get pushed onto disks left, right and centre, wherever there is space, with the pious mental resolve to 'sort them out at the weekend'. Enough said.

### 3.20 Starting Work.

After powering up, certain messages will appear on the screen, ending in this symbol:

A&gt;

This is the 'prompt' sign, so called because the computer is inviting, or prompting you to type in some instructions. Now keep the system disk in the drive. You can now try out some of CP/M's commands.

A word of reassurance. Provided you do not physically handle the keys roughly, whatever you type cannot do any damage. Beginners are often terrified that they might strike a combination of keys, some fearful 'lost chord', that will cause the screen to explode or result in a curl of blue smoke rising from the machine, and the insurance man pointing regretfully to some nasty paragraph in small print. It can't happen. The worst that can befall you is to lose a day's work, or accidentally erase a disk, but having read the next few pages, that would be the result of ordinary carelessness, which is avoidable. As a 'learner driver', rest assured that you have not innocently taken charge of a supercharged Ferrari.

You cannot type text in straightaway. The prompt sign must always be followed by a command. The command may well be to allow you to type in your piece of text, but the command has to come first. As there are many varieties of commands, it is worth a brief word in general on those first:

The command word has to be one that the computer recognizes, but there are not all that many and the full list is set out below. A word of caution is necessary here: The computer will only recognize certain commands if they are on disc. These are called 'Transient' commands because they are not one of the small group of commands that the computer always remembers

```
DIR (see what is on the disc)
ERA (remove a file from the disc)
REN (change the name of a file)
TYPE (type out a text file)
USER (change to another user area)
```

All command instructions to the computer, including filenames, may be entered in either upper or lower case. The computer will always display them in upper case, converting them if necessary. Since it is easier to type in lower case, examples in this manual of commands that the user would type in have been given in lower case. Examples of commands that the computer would display appear in upper case. It may seem a confusing discrepancy, but it is no more than the way things normally are.

The command line will be either a command word on its own, or a command word plus qualifying information.

If you were to type 'dir' in response to the A> prompt, the directory listing would appear as follows:

```
A> dir<cr>
A: P11CPM3  EMS : BANKMAN  BAS : PROFILE  ENG : SUBMIT  COM
A: SEIKES   COM : KEYS      CCP : LANGUAGE COM : SET24X80 COM
A: PALETTE  COM : SETSIO    COM : SETLIST  COM : DISCKIT3 COM
A: DATE     COM : DEVICE    COM : DIR       COM : ED       COM
A: ERASE    COM : GET       COM : PIP       COM : PUT       COM
A: RENAME   COM : SHOW      COM : TYPE      COM : SET       COM
A: SETDEF   COM : AMSDOS    COM : BANKMAN  BIN : KEYS    WP
```

The files that are followed by a COM are 'transients'. As we have

mentioned already, if the disc is changed or removed from the drive, the computer will seem to be suffering from amnesia, except for the handful of commands that the computer always remembers.

The following commands will cause things to happen if simply typed and followed by a carriage return, though they can be persuaded to do other things if you specify them in a command tail:

```
SETPIO    (Amstrad utility) Displays the current setting of the serial port.
AMSDOS    (Amstrad utility) Leave CP/M for AMSDOS
SETDEF    Displays how and in what order drives are searched
SHOW      Displays remaining usable space on each drive
DATE      Displays the date and time
DEVICE    Displays the way the input/output ports are configured
DIR        Displays the list of files on the current drive
```

Others start a sequence, in which a succession of screen displays show easy to follow instructions:

```
HELP      A step by step guide to the utilities
DISCKIT   (Amstrad utility) For copying formatting or verifying disks
DEVICE    Displays input/output port configurations
```

The beginner is urged to try all these - with care. The top list cannot land you in any trouble, as it simply displays information. The second list invites you to alter things. Obviously don't, at first. But the 'no change' option is usually clear enough - it will be either a y (yes), n (no) or <cr>, enabling you to have a good look round and exit safely. You will gain the confidence of the owner of a strange new house who has at least peered into every room, rather than crouched over a paraffin stove (named Wordstar!) in one room only, wondering what terrors the others hold in store. Almost your first action (see 'How to Copy your System Disk') is to make a copy of your Master Disk, so to be doubly sure, make two copies and use one for your investigating. It is a good idea to put a copy of the system disc away somewhere with the write-protect tag in place, just in case you have a nasty accident with your system disc. The most dangerous commands are the DISCKIT and ERA commands. I have seen grown men cry after formatting a disc from which they had neglected to take regular backups, thereby wiping out two months work.

Often the user will be working with bought-in software packages, like Supercalc or Wordstar. These are called special applications programs and they too are invariably activated by a single command and a carriage return.

### 3.21 Changing drives

If the prompt is showing

```
A>
```

and the file you want is on a disk in drive B, you can switch the computer's attention to that drive ('access' drive B) by typing

```
A> b:<cr>
```

and the prompt will change to

```
B>
```



### 3.22 Scrolling

The screen can hold some 2000 characters, but this is not nearly enough to view a long document. The equivalent of using the typewriter roller to roll the page up and down is a procedure like unravelling a medieval scroll in order to view a portion at a time, - hence the expression scrolling the screen.

With some word processing programs, there is complete flexibility to scroll up or down, or even sideways, automatically or manually, as one would expect with a system designed to handle large quantities of text.

With CP/M, the potential is more restricted and is limited to text scrolling in one direction only (upwards - the sense in which it is read) and to a simple stop/start arrangement. It is not to be despised, though. Very often, one wants just to have a quick look at a few files - possibly print out portions of them. It is minutes quicker to execute a TYPE command, with a stop/start scrolling and a start/stop print command, than it is to crank up a word processing program, and spend precious time listening to the machine grunt and groan to bring a file up from the cellars, then another delay while it puts it away again. This is the shortcoming of the w/p program, not your computer, which is faster than most.

The need for scrolling arises in the situation described above. The normal (default) condition, when TYPing a long file, is for the screen to display the text a screenful at a time, requiring a <cr> before displaying the next 23 lines. Exact instructions are given later (see 'How to Print a File'), but this can be overridden so that the file scrolls continuously, and commands to stop, start and print out are effected by ^S, ^Q and ^P, as in the sequence:

```
^S      stop output to the screen)
^P      switch output to the printer as well)
^Q      restart output to the screen)
```

to switch off the printer, then the same sequence is followed:-

```
^S      stop output to the screen)
^P      stop output to the printer
^Q      restart output to the screen)
```

### 3.23 ONCE ROUND THE BLOCK

1. How to format a disk
2. How to copy a disk
3. How to find out what's on a disk
4. How to see what room there is on a disk
5. How to name a file
6. How to print a file
7. How to create a file
8. How to correct and edit a file
9. How to copy a file
10. How to erase a file
11. How to rename a file
12. How to date and time-stamp a file
13. How to run a program

#### Introduction

If a driving school insisted on spending the first few lessons teaching the theory of the internal combustion engine, at the same time assuming that the pupil had had extensive driving experience, it would not attract many customers. A pupil simply wants to get in and drive, and gradually acquire skills as he or she is ready to learn them. Yet both these errors seem to abound in computer manuals, accounting for much of the hostile reaction to computers to be found in many quarters, also the unwillingness to explore beyond the one or two routines learnt at the very beginning.

The title of this section therefore speaks for itself. No attempt will be made to explain whys and wherefores.

Some operations in fact you may never use, such as ED for actually creating a file and making alterations in it. It is much more likely that you will use Wordstar or some other text editor to do this, but we shall include it nevertheless. It has many convenient uses, and having made it work once may tempt you to come back for another look at a later date.

#### 3.23.1. How to format a disk.

With the System Disk in drive A, insert the disk to be formatted in drive B. If the screen reads B> type a: and <cr> so that it appears as A> and type the word DISCKIT3 (DISCKIT on the Amstrad 8256):

```
B> a:<cr>
A> diskkit3<cr>
```

A menu appears on the screen inviting you to either COPY, FORMAT, VERIFY or EXIT. Take a deep breath, and just check that the disc you are about to format is really new or ready for recycling. Then press the f4 function key (not the 4 key as it suggests!). A new menu appears. Choose the data format (press f6). put the disc in the drive, and press either f8 or f5 according to the drive you are using (f8 for the built-in drive, f5 for the extra one)

NB: Formatting is also carried out on disks that have been in use, as it is the most effective way of wiping all data off it, including protected or hidden files. It is therefore essential to check before formatting that there is nothing of importance still remaining on it.

There is another initiation type of procedure, which allows a disk to be date stamped. It may not be so immediately necessary as plain formatting, but while you are still flushed with success from that, you may like to add another scalp to your belt. Type the instruction below and follow the sequence:

A> initdir b:<cr>

INITDIR WILL ACTIVATE TIME STAMPS FOR SPECIFIED DRIVE.

Do you want to re-format the directory on drive: B (Y/N)? y

The prompt A> shows that the task is done and that the diskette is initiated for date stamping. It is then necessary to use the SET command to assign a directory label to the diskette and to select the required date stamp. See section 3.23.12 'How to date and time stamp a file'.

If you have any other blank disks, it's as well to do them all while the routine is set up.

### 3.23.2 How to copy a disk.

Copying your System Disk is a safety precaution. If your dealer has not already done it for you, it is essential you carry out this procedure as soon as possible. System Disks are not specially vulnerable, but mishandling is more likely during the early days of using a computer, and it may be expensive to replace.

In a two-drive system, the easiest way of copying the contents of a disc is to place a formatted, empty disk in drive B: and a disk with PIP in it on drive A: Type

A> pip<cr>

and it will respond with a '\*' sign. Take out the disc from drive A: and replace it with the disc that you want copied; now type

\* b:=a:\*. \*<cr>

and the disc will be copied.

Most computers come with special utilities that do the job rather faster, addressing the hardware directly. These make exact copies of the disc, whereas PIP takes the opportunity to tidy up the way each file is actually stored on disc. Amstrad CP/M 2.2 computers come with two different versions, depending on whether you have a single or twin drive system. CP/M Plus uses DISCKIT to do an exact copy of a disc.

A> diskkit3<cr>

A menu appears on the screen inviting you to either COPY, FORMAT, VERIFY or EXIT. Please check that the disc you are about to copy onto is really new or ready for reuse, as all existing data on it will be lost. Press the f7 function key (not the 7 key that it invites you to press!). A new menu appears. Choose the drive to copy from in the next menu (press either f8 to copy from the built-in drive, or f5 to copy from the extra one). The next menu appears. Choose the drive to copy to (press either f9 to copy to the built-in drive, or f6 to copy to the extra one). Now follow the instructions that appear on the

screen.

### 3.23.3 How to find out what's on the disk.

A freshly formatted disk will be blank, but if you have a disk and want to find out what's on it, insert it in the drive.

Type DIR in response to the A> prompt. Follow this with the name of the drive and a colon, if it is not the current one, and a <cr>

```
A> dir b:<cr>
```

and the directory will be displayed.

### 3.23.4 How to see what room there is on the disk.

To work away entering data all morning, only to find there isn't enough room on the disk to store it, is the sort of mistake one tends to make only once! The file SHOW.COM must be on disk. On a two-drive system, the system disk could be in drive A and the disk to be investigated in drive B. Type show b: and <cr> It doesn't matter if the prompt is showing A> or B>

```
A> show b:<cr>
```

and the number that appears, followed by a k, reveals roughly how many thousand spaces there are left for characters on the disk. Type ^C to get back to the prompt.

To find out just how much space each file is taking up, after the normal directory instruction, type the word 'full' in square brackets:

```
A> dir [full]<cr>
```

Directory For Drive B: User 0

Name	Bytes	Recs	Attributes	Name	Bytes	Recs	Attributes
BASALT	BAK	26k	208 Dir RW	BASALT	II4	28k	218 Dir RW
BASALT	INT	4k	27 Dir RW	CIRCLE	ASM	2k	3 Dir RW
CONSOLE	DOC	40k	313 Dir RW	CONKEYS	EX	2k	15 Dir RW
CPMG	DOC	94k	661 Dir RW	DIRECTOR	EX	2k	3 Dir RW
FILEPROC	EX	6k	43 Dir RW	POLICY	DOC	2k	8 Dir RW
OVERVIEW	DOC	4k	17 Dir RW	INVTIMNT	DOC	36k	281 Dir RW
RECT	ASM	2k	3 Dir RW	DHSS1	LTR	4k	19 Dir RW
RULE	ASM	2k	3 Dir RW	SECTION2	TXT	2k	5 Dir RW

Total Bytes = 256k Total Records = 1892 Files Found = 18

Total 1k Blocks = 246 Used/Max Dir Entries For Drive B: 22/ 64

### 3.23.5 How to name a file

Like files in a filing cabinet, each one must be named distinctively, ('unambiguously') so that there is no confusion. Eight letters (or numbers) are allowed for a filename, then another three, separated by a full stop, for a further identification tag, usually abbreviated - eg '.DOC' for document. This

tail is called the filetype.

Do's and don't's: certain characters may never be used in any part of the labelling, or the computer will object strongly. These are:

< > . , ; : = [ ]

Two other characters, ? and \*, may be used but in a special way (see the next chapter)

There are also certain abbreviations reserved for other functions that ought not to be used, namely:

.\$\$\$ .ASM .BAS .COM .HLP .SUB

so if you are in the habit of dealing with Assistant Scoutmasters or the local Communist party, as you may well be, it is best not to tag their files with the obvious abbreviations.

The letters allowed do not have to be taken up, and there does not have to be a filetype tagged on the end. Some permissible labels are therefore:

jones  
jones.ltr  
jones3.ltr  
payroll.aug  
nwreport.4a

### 3.23.6 How to print a file

To be able to print a file, one must first know how to refer to it, in order to give the right instruction. As was explained in 4 above, the full title of a file is a name, a full stop and then a three letter abbreviation to show the type, eg a text file for Sandersons might be named SANDRSON.TXT, but it would appear in the directory (see above) without the full stop, and with a gap separating the two parts of the title, eg SANDRSON TXT. To handle the file in any way, including giving instructions to print it out, the dot must be put back in and the letters run together.

Insert the disk in the drive, and check that the file exists on disk by calling up the directory (see above).

In computer language, 'print' means simply to make it appear on the screen; in normal speech 'print' means to make appear in black and white, on paper. We shall deal with both senses of the word.

#### (i) How to make it appear on the screen

Simply to display a file on the screen, to see what you've got, just key in the word 'type' and the filename, followed by <cr>:

**B> type sandrson.txt<cr>**

and the file will be displayed, a screenful at a time. To get back to the prompt, type ^C.



(ii) How to print it out on paper

Method 1.

For a paper (hard copy) print out, proceed much as before: key in the word 'type' and the filename, but this time add the expression [no page] and ^P before the usual <cr>. (The [no page] directive prevents the words 'press carriage return to continue' appearing every 23 lines)

```
A> type sandrson.txt [no page]^P<cr>
```

The printer will continue to print out anything that appears on the screen, until you type another ^P to switch the printer off again.

Method 2.

This is neater, but involves a more complex looking command, though there is no need to press ^P. Make sure you are logged on to the correct drive, then simply type:

```
B> pip lst:=sandrson.txt<cr>
```

### 3.23.7 How to create a file

Normally this would be done with a text editing program such as Wordstar, but as this is not often included with the standard CP/M software, we will introduce the CP/M editor ED.

Type ed, a space, the name of your file and <cr> See 4. above for allowable filenames - A \* prompt will appear:

```
B> ed add65<cr>
NEW FILE
*
```

Type vi and <cr> This will enable you to view the number of each line and start inserting text. Type your lines carefully, using backspaces for deletions, and <cr>'s as normal. <cr>'s can be repeated all the way back to the beginning, if needed. Better that, for a new recruit, than have to edit out the mistakes later. At the end, type ^Z, then e to end (and save) the file.

```
A> ed add65<cr>
: *vi<cr>
1: MML Systems Ltd.<cr>
2: Wholesale Fruiterers<cr>
3: Contact: Geoffrey Smart<cr>
4: 17 Cooper Street<cr>
5: London WC1A 3TY<cr>
6: 01-580-3432<cr>
7: ^Z
: *e<cr>
```

Check the contents in the normal way by using the type command (see 5 above) and your effort should appear as you typed it, but this time less line numbers and editing commands:

```
A> type add65<cr>
MML Systems Ltd.
Wholesale Fruiterers
Contact: Geoffrey Smart
17 Cooper Street
London WC1A 3TY
01-580-3432
```

If it is correct, all is well; if not, read on.

### 3.23.8 How to correct and edit a file.

Suppose the file was quite correct, but the contact's name and phone number have changed and need altering. We shall not explain how or why but simply describe the slow but sure way of achieving this: get into the file, locate the line containing the error, kill the whole line, retype it correctly and exit. Repeat for the second line to be edited.

Start as before, except that this time NEW FILE will not be displayed. When the prompt : \* appears, type `EAET` and `<cr>`. This will cause all (E) the file to be Appended, ie brought out of memory to be available for working on, also all (E) the file to be Typed out on screen, with line numbers. It will end with the prompt 1: \*

```
A> ed add65<cr>
: *EAET<cr>
1: MML Systems Ltd.
2: Wholesale Fruiterers
3: Contact: Geoffrey Smart
4: 17 Cooper Street
5: London WC1A 3TY
6: 01-580-3432
1: *
```

The prompt is inviting you to edit line 1. We are interested first of all in line 3, which is 2 lines down, where we shall kill the line and insert a replacement, including the `<cr>`. So, at the \* prompt, type `2lki` and the new entry as follows:

```
1: *2lkiContact: Jennifer Dyer<cr>
4: *
```

The prompt now invites an edit of line 4 (the next line), which we decline by typing `e` for end and `<cr>`.

```
4: *e<cr>
```

The partly edited file is now called up again, and a similar procedure amends the telephone number in line 6, which is 5 lines from the top:

```
A> ed add65<cr>
: *EAT<cr>
1: MML Systems Ltd.
2: Wholesale Fruiterers
3: Contact: Jennifer Dyer
4: 17 Cooper Street
5: London WC1A 3TY
6: 01-580-3432
1: *51k101-580-7679<cr>
7: *e<cr>
```

To verify, bring the file onto the screen in the normal way with TYPE, as above.

### 3.23.9 How to copy a file

Making copies of files is one of the best ways of preserving your sanity when working on a micro. Even in the best ordered of lives, things can go wrong with a file you are working on, and the authors of this book are too well familiar with the awful sinking feeling when one finds that the file one has spent a week of work on has been accidentally erased, and there are no copies of it.

PIP is the utility that is used to copy a file. Typing:-

```
pip
```

should result in the signon message:-

```
CP/M 3 PIP VERSION 3.0
```

```
*
```

If you just get:-

```
PIP?
```

Then you have not got PIP.COM in the current drive. If this is the case, then put a disk in which has PIP.COM on it and try again.

To make a copy of a file on the same disk is now easy. You must think of a new name for the copy because, for obvious reasons, CP/M does not allow two files with the same name on the same disk. If you are copying the file 'SECRETS', then you might want to call the new file 'SECRETS.BAK'. Type:-

```
secrets.bak=secrets
```

Note that the new file comes first. The disk should make a whirring sound and the '\*' prompt should reappear. When it does, pressing the carriage-return key will return you to CP/M. Certain things can go wrong. If you get the order the wrong way round, or misspell the filename of the file you want copied, then the message:-

```
ERROR: FILE NOT FOUND -xxxx
```

will appear. Also there could be no more room on the disk, or, more unlikely, no more directory space, in which case you will be told.

If you want to make a copy of a file onto another disk, then you must specify the disk. eg:

```
m:secrets.bak=secrets
```

will copy the file 'SECRETS' to a file on m: drive called 'SECRETS.BAK'. If you are copying onto another disk, then there is no necessity to change the name of the file, in which case the shorthand form:-

```
m:=secrets
```

would do. If you wanted to copy the file from drive A: to drive B:, then the instruction:

```
b:=a:secrets
```

would do the trick.

PIP.COM has a lot more tricks up its sleeve and these are explained in much more detail in chapter 4.

### 3.23.10 How to erase a file

Before the file can be erased, it must be correctly named. See above. With the disk containing the file for erasure in the drive and with A> or B> showing on the screen, check the file's existence by calling up the directory (See 2 above). If it is there, type 'era' (or 'erase'), followed by the filename and a <cr>:

```
B> era sandison.txt<cr>
```

### 3.23.11 How to rename a file

As above, check the required file is there, on the disk in the drive, and that the B> prompt is showing. Then follows what looks like an equation, with the new name first:

```
B> ren matthews.doc-sandison.txt<cr>
```

As above, type 'rename' or simply 'ren'.

### 3.23.12 How to date and time stamp a file.

When recording price lists or inventories, it is essential to know when the file was either (i) created, (ii) last looked at or (iii) last updated. Unfortunately it is impossible to know all three, but the last update can always be seen, and either one of the first two. Here's how:

First the directory itself must be prepared so that it can accept date stamping. This is a once and for all operation for each disk and is best done whenever you format a diskette (see 3.23.2 How to format a disk). You use the utility INITDIR to do this. (you just put in the system disc and type INITDIR. If you only have a one-drive system, take it out and put in the disc you want to prepare. INITDIR will talk you through the procedure.)

Then the required combination - either <create + update> or <access + update> - must be (precisely) specified. We shall choose the latter:

```
A> set b:[access-on,update-on]<cr>
```

Directory Label	Passwds Reqd	Stamp Create	Stamp Access	Stamp Update
B:LABEL .	off	off	on	on

Once date stamping is enabled, then the directory entry for each file will be updated with a new date stamp on subsequent Create, Access and/or Update. The date stamp is taken from the BDOS clock. In some computers this clock is connected to an on-board real time clock with battery backup, but as the Amstrad computers are not supplied with a real time clock, the time and date must be entered whenever the operating system is loaded. Otherwise, the date stamp recorded on the file directory will show a date in 1978 which is not very helpful and defeats any purpose in enabling the date stamping. The best method of ensuring the bdos clock is correctly set is to add the transient command

'date s' to the file PROFILE.SUB on each system disk. This file PROFILE.SUB is automatically run whenever the operating system is loaded. By adding 'date s' to this file, the user will be asked to enter the date and time at the start of every session.

To find all about a disk, one always types 'dir [full]', but from now on it will reveal date and time information as well. In a few days' time, this command might yield:

```
A> dir b:[full]<cr>
```

Name	Bytes	Recs	Attributes	Prot	Update	Access
JOBSPEC	DOC	84k	661 Dir RW	None	05/12/85 09:09	
ANNUAL	RPT	2k	2 Dir RW	None	05/13/85 14:39	
MD32	LIR	40k	313 Dir RW	None		
ACCOUNTS	MAY	22k	172 Dir RW	None		
ITT	DOC	26k	193 Dir RW	None		
MD33	LIR	0k	0 Dir RW	None		
MERGER	DOC	36k	281 Dir RW	None		05/10/85 11:09
PERSONNL	TXT	14k	97 Dir RW	None	05/10/85 12:10	05/10/85 11:10
Total Bytes = 224k Total Records = 1719 Files Found = 8						
Total 1k Blocks = 220 Used/Max Dir Entries For Drive B: 18/ 64						

This reveals that the disk was date stamped after files 3 - 7 had already been written; that files 1 and two were written on the 12th and 13th May respectively; that the date of origin of file 8 is unknown (we have decided it does not need to be known), but it was updated late in the morning of the 10th and finally that the operator took a quick look at file 7 just before that, but didn't alter anything.

Unfortunately the method of date stamping used by CP/M Plus is not compatible with MP/M II, and disks which are to be interchanged with MP/M II or the 16 bit operating systems DOS Plus, C-CP/M and C-DOS should not be initialised for date stamping.

### 3.23.13 How to run a program.

Programs are just like other files but have the filetype '.COM' (eg PAYROLL.COM). If the program file is on disc, then you run it just by typing its name at the system prompt (A>) You do not need to type in the '.COM'! If you cannot remember the name of the program, or you think you may have made a mistake, then type

```
A> dir *.com<cr>
```

and all the available programs will be displayed on the screen (If any COM files have the System attribute, then repeat the command replacing 'dir' with 'dirs'). To run SUPERCALC, for example just type SC at the system prompt eg:-

```
A> SC<cr>
```

If you want to do more than these basic tasks, then it is time to tackle the next chapter, which describes the transient commands rather than the tasks.



## CHAPTER 4 – The Console Command Processor

### The Console Commands.

CP/M Plus (CP/M v3.1) is a greatly enhanced version of CP/M and has several highly desirable features that do not exist in previous versions. The Transient commands reflect the operating system's greater sophistication. The extra facilities, such as password protection, or time/date stamping need not be used, but the power is there if you really need it. If you are familiar with CP/M 2.2, as on the Amstrad 464/664, you will find that CP/M Plus offers

- (1) Faster disk work due to directory and data buffering.
- (2) Improved utility files.
- (3) On-screen Help texts.
- (4) Multiple commands on a line for batch work
- (5) Easier control over I/O devices
- (6) Faster program load
- (7) Record of date of creation and access on files
- (8) Automatic backup of altered files via PIP
- (9) Improved software development tools
- (10) User control over disk assignment and file searches
- (11) Easier use of batch processing facilities
- (12) Assignment of console i/o to file
- (13) Automatic disk login
- (14) Automatic extension to the system (eg. for graphics)
- (15) Improved error trapping
- (16) Larger transient program area

When you are interacting with CP/M, what you are really dealing with is a small part, the Console Command Processor. CP/M is, logically speaking, divided into three parts; The BIOS (Basic Input/Output System), The BDOS (Basic Disk Operating System) and the CCP (Console Command Processor). The average user only interacts with the CCP. The BDOS and BIOS are for programmers only. This chapter deals with the way that the User interacts with the operating system and therefore addresses itself mainly to the CCP and the Transient commands. For information on the BDOS and BIOS, see elsewhere in the book.

The CP/M CCP is designed to be used by people who are reasonably educated in the use of computers. It is not intended for beginners. 'User-friendliness' is a boon to the beginner, but it gets in the way as the user becomes more and more familiar with what he is doing. The CCP is intended to require as few keystrokes as possible for the user to get the effect he/she wants. The beginner or turnkey system user need never see the CCP at all, and the CP/M Plus CCP is designed to be replaceable by a user-friendly shell for special applications.

#### 4.1 On Starting

When the system prompt appears, then the computer is ready to accept commands. The system prompt will look something like this :-

A>

or it may consist of an upper case letter and a digit along with the prompt like this :-

1B>

This prompt sign indicates that the computer is waiting to be given a command to do something. The command must first be one that the computer recognises - that is, it must feature on the discs in the drive(s). Also the syntax must be right: correct order and correct spelling. Computers have no subtlety in trying to work out what you think you are trying to tell it.

A CP/M command line consists of a command word followed by an optional command tail that might consist of drive specifications, file specifications, strings, options or parameters. The command word itself is usually the name of a transient utility stored on disk.

Typically, the user will only need to type the name of the application package in question. For example:-

A> payroll<cr>

--the user has, in fact, typed a command, which tells the CCP to load the transient program called PAYROLL.COM

Should you make a typing error, it is possible to make corrections. The following keystrokes should cover most eventualities:

<bs>, <del> and ^H all backspace, deleting the last character.  
 ^A backspaces without deleting (moves the cursor only).  
 ^F moves the cursor forward (without pushing the whole line to the right).  
 ^G deletes the character under the cursor.  
 ^X deletes the whole line to the left of the cursor.  
 ^K deletes the whole line to the right of the cursor.  
 ^B moves the cursor to the beginning or to the end of the line.

(Insert mode is always in operation, so characters and spaces in the middle of the line always push the rest of the line along).

The user can type a batch of commands at the console and these are then executed in sequence. For example:-

1A> show ! ws programe.asm ! mac programe ! link programe ! programe<cr>

This is one of the most welcome added features of CP/M Plus. No longer does the CCP have to execute commands singly - while the operator waits helplessly until each is finished. Commands can be 'queued' and the whole batch set in motion, which gives the operator time to tackle other work until the computer is finished.

Commands, however dissimilar, may be queued by use of the symbol ! for example:

A> era b:ufn1:pip m:=ufn2:pip lst:=b:milist.mdx(nz)!ren b:ufn2=b:ufn3<cr>

The output to the screen in CP/M Plus can be controlled by pressing the two keys Control S (^S) and Control Q (^Q). The ^S key temporarily stops more characters going onto the screen (stops output) and the ^Q key restarts it.

If, at any time, you wish to put output onto the printer as well as the screen, then the Control P (^P) key is used. This can be done at any time by the sequence:-

```
^S.....(stop output to the screen)
^P.....(switch output to the printer as well)
^Q.....(restart output to the screen)
```

to switch off the printer, then the same sequence is followed:-

```
^S.....(stop output to the screen)
^P.....(terminate output to the printer)
^Q.....(restart output to the screen)
```

## 4.2 The console commands

Within the CCP (console command processor) are 6 built-in commands. Other utilities, or transient commands, of CP/M are supplied as separate programs indistinguishable from application programs. The transient command is loaded from the disk. For the CCP to load a transient, it must be on a disc placed in a drive. Application programs (or Transients) include the utilities that are supplied by Digital Research but will include such application packages as SUPERCALC or WORDSTAR. These would need to be present on disk as SC.COM or WS.COM and would be invoked by typing SC or WS in response to the system prompt. (eg:- A>)

The built in commands consist of DIR (List file names in a directory), DIRSYS (Display the 'system' files), ERASE (Erase specified files), RENAME (Rename the specified file), TYPE (Type the contents of a file on the console) and USER (Change the currently logged user (CP/M 2)). DIR, ERASE and TYPE all have extended versions for more complex tasks that are loaded from disc.

The transient commands (that have to be loaded from disc) provided with the system consist of COPYSYS (Create a new system disk), DATE (set or display the date and time), DEVICE (alter device configurations), DUMP (Display a file in ASCII and HEX), ED (Create or alter a text file), GET (get console input from a file rather than the keyboard), HELP (Display information on using CP/M), HEXCOM (convert a '.HEX' file into a '.COM' file), INITDIR (allow date stamping on disk files), LINK (link '.REL' files together to make a '.COM' file), MAC (assemble a program), PIP (copy, combine, or transmit files), PUT (put console output into a disk file), RMAC (assemble a program into a '.REL' module), SET (set file options), SETDEF (Set system options), SHOW (Display disk and drive statistics), SID (debug a program), SUBMIT (execute several commands together as a batch) and, XREF (cross reference a program source).

Within the CCP, File references can refer to a single file by name (eg FILENAME.TYP) and are described as 'unambiguous' (ufn) or they can refer to a class or set of filenames (eg \*.MAC): These are referred to as 'ambiguous' (afn). CP/M Plus identifies every file by its unique file specification, which can consist of four parts: the drive specification, the filename, the filetype and the password. A file specification ("filespec") is any valid combination of the four parts of a file specification, all separated by their appropriate delimiters. A colon must follow a drive letter. A dot must precede a filetype. A semicolon must precede a password.

This chapter will refer to the following parts of the file specification as a shorthand :-

d:	drive specification	single letter (A-P). If omitted the current drive is used.
filename	file name	1 to 8 letters and/or numbers. If omitted a blank name is used.
typ	file type	1 to 3 letters and/or numbers. If omitted a blank type is used.
password	password	1 to 8 letters and/or numbers. If omitted the default password is used if the filename requires a password (see SET [DEFAULT= ]).
afn	Ambiguous file name	A file specification that optionally can be ambiguous and can specify the drive.
ufn	Unambiguous File name	A file specification that must be unambiguous and can specify the drive.

Optional parameters are shown within curly brackets (...). These should not be typed. Many CP/M commands use square brackets [....] to denote options and these must be typed.

Valid combinations of the elements of a CP/M Plus file specification are:-

```
filename
d:filename
filename.typ
d:filename.typ
filename;password
d:filename;password
filename.typ;password
d:filename.typ;password
```

If you do not include a drive specifier, CP/M Plus automatically uses the default drive.

Some CP/M Plus commands accept wildcard (\*) and (?) characters in the filename and/or filetype parts of the command tail. A wildcard in the command line can refer to many matching files in one reference on a particular user number and drive. Ambiguous file specifications (AFN) contain wildcard letters (\*) and (?) whereas unambiguous ones (UFN) do not.

An unambiguous file reference is an exact name of the specified file. It consists of up to eight characters in the file name and up to three characters in the file type, with an optional drive specifier and password.

The characters used in specifying an unambiguous file reference should not contain any of these characters:-

```
< > . , ; : = ? * [ ]
```

The form of an ambiguous file reference is similar to an unambiguous reference, except that the symbol '?' may be interspersed throughout the file reference. In various commands throughout CP/M, the '?' symbol matches any

character of a file name in that position in the name, e.g. P??ASM will refer to PAT.ASM, POL.ASM, PUW.ASM etc. The '\*' symbol is used to refer to all characters of a file name or file types so that, for example, \*.\* would refer to all the files on the active disk. W\*.\* would refer to all files beginning with W on the disk whereas WS\*.OVL might refer to all the Wordstar overlay files. Unless referring to files on the active disk (eg, when B> appears, then drive B is active) file references should indicate the drive followed by a colon at the start of the file reference.(eg B:CPM.ASM)

#### 4.3 The Editing Keys

When writing a command line on the console, the following keys do editing and control functions:-

<u>chara</u>	<u>function</u>	ASCII <u>code</u>
^A	Move the cursor one character left	01H
^B	Move cursor between beginning and end of command line	02H
^C	Reboot CP/M (warm boot)	03H
^E	Start a new line	05H
^F	Move the cursor one character right	06H
^G	Deletes the character under the cursor	07H
^H	Backspace and delete	08H
^I	Tab 8 columns	09H
^J	Line feed	0AH
^K	Deletes to the end of the line	0BH
^M	Carriage return	0DH
^P	Printer on/printer off	10H
^Q	Restarts display output after a ^S	11H
^R	Retype current line	12H
^S	Stop display output - ^Q starts it again	13H
^U	Delete line, updates with the characters to the left	15H
^W	Recalls the previous command line if no command present	17H
^X	Delete to the beginning of the line	18H
delete	rubout last character	7FH
Backspace	rub out the last character (see ^H)	08H

#### 4.4 File Protection

In real business use, it is vital to protect sensitive and confidential information contained in files from unauthorised use. CP/M Plus features file protection by means of passwords and the logging of access by means of time and date stamping. Files can also be protected from accidental erasure by making them Read-only. An entire disk can be made Read-only. The SET utility manages these features

Passwords are also managed by SET. Both programs and data files can be made secure with passwords and SET itself can also have a password applied to it in order to prevent passwords being altered once they are set.

#### 4.5 User Areas.

CP/M Plus is a single user system, in that only one user can use a CP/M Plus system at any one time. However, the advent of large-capacity hard disks has made the USER feature of CP/M Plus more important. The USER feature enables a disk to be divided into 16 areas. In any USER area, the operator can

access system files in USER 0 (the default user area) but can keep his own files separate. This is useful in two circumstances, where more than one person uses a disk, or if the user wishes to keep tasks or programs separate. If the transients are all set to be 'system' files, by means of the 'SET' transient, then they can be accessed from the other user areas. Note that CP/M 2.2 user areas were almost unworkable, due to the need to have copies of the transient commands on every user area that was actually utilised.

The current user area can be seen immediately in the system prompt as follows

```
1A>.....This means you are on Drive A, and user 1
12C>.....This means you are on Drive C, and user 12
4B>.....This means you are on Drive B, and user 4
A>.....This means you are on Drive A, and user 0
M>.....This means you are on Drive M, and user 0
```

User numbers allow more precise categorising and subdividing of file directories and are almost essential on large-capacity drives such as hard discs. They work similarly to the sales assistants' buttons on electronic cash tills, whereby assistants first punch in their own number before entering a transaction, so that their own personal takings (or errors) can be subtalled at the end of the day. With CP/M Plus, a number may be inserted before the drive letter, if for instance there are several operators having access to the same disk and each wants to keep their own 'library' of files distinct. Operator 4 would therefore begin all operations with:

```
A> user 4<cr>
```

resulting in the prompt:

```
4A>
```

or, perhaps more directly:

```
A> 4:<cr>
```

```
4A>
```

or:

```
A> 4b:<cr>
```

```
4b>
```

The maximum number allowable is 15. Mostly, commands will only operate on files in that User Number category, thus:

```
7b> dir<cr>
```

will display only those files in the User Number 7 directory. However, the transient utility PIP can be used to transfer files in different user numbers and SHOW can display statistics on user areas.

The User Number device is especially useful for hard disks, which have an enormous capacity, and where the more subdividing that can be achieved the better. To avoid the duplication of all the transient command files to each user level, the transient commands need only reside on User Number 0, but with the System attribute.



Setting the System (SYS) attribute for the transient command files on user 0 with the SET command type:

```
A> set *.com[sys]<cr>
```

will this will set the SYS attribute for all the '.COM' files residing on drive A:, User 0. Then, assuming that the file 'pip.com' is not duplicated onto User 7, this transient command can still be entered:

```
A> user 7<cr>
7A> pip<cr>
CP/M 3 PIP VERSION 3.0
*
```

Some transient commands use additional files to the '.COM' file. Take for example the wordprocessor program Wordstar, this consists of a file of type '.COM' and a number of files of type '.OVR'. For successful use from different user areas, not only must the '.COM' file on user 0 be set to the SYS attribute, but all the '.OVR' must also be set to the SYS attribute, otherwise after successfully loading Wordstar, the program will halt as it is unable to find the '.OVR' files.

#### 4.6 The Resident Commands.

In CP/M, there are Resident and Transient commands, which in many ways are similar to a permanent and a casual workforce on a building site. No builder today has many on his actual payroll. He usually operates with very few permanent staff and calls up electricians, plasterers and the like only when needed. That is just the distinction between the Resident and the Transient commands.

The five resident commands, which are always 'on site' are the commands that direct the USER to a numbered area of the disk, tell him the DIRectory of what's on it, ERase or REName a file, and finally TYPE it out onto the screen or printer. No matter which disk drive you are on, these commands will always work. Remember that, in certain circumstances, the task proves too much for the resident version of a command and a transient version of the command is brought off the disc. For clarity we document both versions of ERASE RENAME and TYPE under the built-in version. As DIR (with options) is so much more complex than the simple CP/M 2.2-compatible version, we will describe both versions seperately.

## 4.6.1 ERASE -To delete files-

Built in commands:

ERA afn

ERASE afn

Transient commands:

ERASE (afn)[CONFIRM]

ERASE

The ERASE (erase) command will remove one or more unwanted files from disk that are specified by afn (Ambiguous File Name). The ERASE command may be abbreviated to ERA. Wildcard characters are accepted in the filespec. Directory and data space are automatically reclaimed for later use by another file. The ERASE command can be abbreviated to ERA. Erase is like a hand grenade - simple to use and highly dangerous. One moment's inattention and you can destroy a week's work in a second. However, clearing the dead wood out of disks is essential to keep them compact and efficient, so when the time comes to dispense with that sales report, the procedure is simple:

```
A> era s/report.jun<cr>
```

Often there is more of a spring clean needed. All the June data has to go (perhaps having been transferred to another disk). So the command becomes:

```
A> era *.jun<cr>
```

The computer knows that this is a fairly drastic act, so it tugs your sleeve:

```
ERASE *.JUN (Y/N)?
```

inviting you to confirm by typing y, or hurriedly retract with an n.

You may want to erase most, but not all, of the June files; in which case tell the computer to ask each time:

```
A> era *.jun [confirm]<cr>
ERASE STOCK.JUN (Y/N)? y
ERASE DIARY.JUN (Y/N)? y
ERASE ACCTS.JUN (Y/N)? n
ERASE ENGS.JUN (Y/N)? n
```

To clear a whole disk (which is quicker than re-formatting it), the command should be obvious:

```
A> era *.*<cr>
ERASE *.* (Y/N)? y
```

The [CONFIRM] option that we have just illustrated informs the system to prompt for verification before erasing each file that matches the filespec. [CONFIRM] can be abbreviated to [C]. ERASE with options is actually a transient, but this is invisible to the user as long as ERASE.COM is on the system disk.

Examples:-

ERA	ERASE prompts for input.
ERA filename.typ	Erase named file on the current drive.
ERA *.*	Erase all files on the current drive (current user).
ERA *.typ	Erase all files of named type on the current drive (current user).
ERA d:filename.typ	Erase named file on the designated drive (current user).
ERASE filename.*	Erase all types of named file on the current drive.
ERA d:X*.*[CONFIRM]	Each file on drive A with a filename that begins with X is displayed with a question mark for confirmation. Type Y to erase the file displayed, N to keep the file.
ERA	The transient utility will prompt for file specification

## 4.6.2 DIR and DIRSYS -To see what is on disc-

Built in commands:

DIR (d:)

DIR (afn)

DIRSYS (d:)

DIRSYS (afn)

DIRS (d:)

DIRS (afn)

Transient commands:

DIR (d:)[options]

DIR (afn)[options]

DIR (afn)[options] (afn[options])

This must be the most commonly used directive. If you are in drive A and want to know what's in it, for instance, type DIR and press <cr>. It will be displayed:

A> dir<cr>

```
A: AMSDOS  COM : SID      COM : DUMP   COM : GENCOM  COM
A: HEXCOM  COM : HIST    UTIL : INTDIR  COM : LIB    COM
A: LINK    COM : MAC     COM : PATCH  COM : RMAC    COM
A: SAVE    COM : TRACE   UTIL : XREF   COM : DD-IMP1  PRL
A: DSHINWA PRL : DDHP7470 PRL : ASM    COM
```

To find out what's on next door in drive B, or in the portakabin 'drive' M, type b: or m:, press <cr> and repeat:

A> m:<cr>

M> dir<cr>

```
M: DIR      COM : PIP     COM : SHOW   COM : ERASE   COM : PUT
M: SYSIN56 $$$ : SYSIN55 $$$
```

Another way to find out the same information is to type 'dir' and then specify 'drive' M. The directory for 'RAM-disc' will be displayed, but you will still be operating from drive A:

A> dir m:<cr>

(display of filenames as before)

The DIR (directory) command causes the names of all specified files and their characteristics to be listed in five columns. The drive name may be specified. All files are listed if no afn (Ambiguous File Name) is given. The DIR command has three distinct references: DIR, DIRSYS, and DIR with Options. DIR and DIRSYS are built-in utilities. The DIRSYS command can be abbreviated to DIRS. DIR with Options is a transient utility and must be loaded into memory from the disk.

The DIR and DIRSYS Built-in commands display the names of files cataloged in the directory of an on-line disk. DIR lists the names of files in the current user number that have the Directory (DIR) attribute. DIRSYS lists the names of files in the current user number that have the System (SYS) attribute. The default attribute is the Directory (DIR) attribute, but may be changed using the SET command. The SYS attribute is a useful method of 'hiding' files from the built-in DIR command, but it can also be used to enable a file held

in user area 0 to be accessed from other user areas. For example the CCP will search files on User 0 with the SYS attribute when searching for a transient file name from a different user area when it is unable to find the filename in the current user area.. Both DIR and DIRSYS accepts the \* and ? wildcards in the file specification.

#### Examples:-

In the following examples of built in commands, the display only includes files for the current user and for files with the directory attribute:-

DIR	Display the file directory on the current drive.
DIR d:	Display the file directory on the designated drive.
DIR *.typ	Display all files of named type on the current drive.
DIR filename.*	Display all types of the designated filename.
DIR x*.*	Display all filenames beginning with the letter x.
DIR x????.*	Display all filenames 5 characters long and start with the letter x.
DIR XY*.FIL	Display all files of type .FIL starting with the letters XY.

In the following examples of built in commands, the display only includes files for the current user and for files with the system (SYS) attribute:-

DIRSYS	Displays all files for the current user on the current drive that have the system (SYS) attribute.
DIRS *.COM	Displays all SYS files with filetype COM on the current drive and user.

The DIR with options is a transient command and will be dealt with later in this chapter. It has to be on disc. If the drive is specified when DIR is invoked (eg A:DIR) then the transient version of DIR is run. It is equivalent to the transient SDIR in MP/M II, Concurrent DOS and DOS Plus.

## 4.6.3 RENAME -changing the name of a file-

Built in commands:

REN new\_ufn=old\_ufn

RENAME new\_ufn=old\_ufn

Transient commands:

RENAME

RENAME new\_afn=old\_afn

The RENAME command changes the names of files on a disk. The file satisfying old\_afn is changed to new\_afn. (Both file specifications must refer to the same drive name). Essentially RENAME lets you change the name of a file, or a group of files in the directory of a disk. To change several filenames in one command use the \* or ? wildcards in the file specifications. The RENAME built command can be abbreviated REN. If you type REN by itself, then the system prompts you for input. However, the command is usually followed by the two names involved. Although the new name can hardly be called the destination, the old name is certainly the source, so the convention of putting that name last is logical:

A> ren newfile=oldfile<cr>

If spaces are inserted (newfile = oldfile) the syntax is not destroyed.

Wildcards \* and ? may be used to hasten proceedings, but they must match exactly the previous pattern. Supposing your letter files are a mixture of '.LET' and '.LTR', the directory may be tidied up by typing:

A> ren \*.ltr=\*.let<cr>

to make them all of a '.LTR' format, but the following would not be allowed:

A> ren \*.ltr=?????let.\*<cr>

Nor would a new name be allowed if a file already exists of that name, as it might if you had created a new file by copying the old one. That is why in this situation there is an invitation (if the transient utility is available) to delete the existing file completely:

A> ren wkinhand.mar=wwinhand.feb<cr>

Not renamed: wkinhand.mar already exists delete (Y/N)?y

For example:-

RENAME	The system prompts for the filespecs:
Enter New Name: filename.typ	
Enter Old Name: oldname.typ	
	File oldname.typ is renamed to newname.typ on the current drive
RENAME *.DOC=*.BAK	Renames all the '.BAK' files to files with '.DOC' filetypes
REN newname.typ=oldname.typ	REName file on the current drive.
REN d:newname.typ=oldname.typ	REName oldname.typ to newname.typ on drive specified by d.

## 4.6.4 TYPE -Display the contents of a file

Built in commands:

TYPE ufn

Transient commands:

TYPE

TYPE afn

TYPE afn [PAGE]

TYPE afn [NO PAGE]

The TYPE command shows the contents of a file or a group of files. The TYPE command displays the contents of the ASCII file ufn on the currently logged-in disk at the console. Tabs are expanded. If ^P is pressed first, the file will go to the printer as well. ^S temporarily halts the action whereas ^Q will restart it.

If the TYPE command is followed by the option [PAGE], it displays the specified file one page at a time, if followed by [NOPAGE], it continuously displays the specified file. The [PAGE] and [NO PAGE] option overrides the default PAGE mode. The transient utility SETDEF can be used to enable or disable the default page mode - see section 4.7.16.

A> type ufn<cr>

would cause the contents of that file to be displayed on the screen until one screen full is displayed (if the default PAGE mode is ON) when the display pauses with a prompt. Pressing a <cr> would bring up another screenful. To cause it to go on scrolling continuously, cancel the page mode (which is the default condition) by either changing the default using SETDEF [NO PAGE], or by adding the [NO PAGE] option to the command page display by entering:

A> type ufn [no page]<cr>

One can scan the file well enough for most purposes like this, but to stop the scrolling for closer inspection, type ^S. To set it going again, press ^Q. To abandon at any time and return to the prompt, type ^C.

Often one requires a hard copy print-out at the same time. Making sure the prompt is displayed, first press ^P and then CR (NB: nothing will show for your efforts on the screen, but the printer will have been activated and may well have given a telltale squeak to prove it). Then enter the TYPE command as before and the print-out will keep pace with the screen display. After the print-out is complete, type ^P again to send the printer back to sleep.

Perhaps you only want an occasional paragraph actually printed out. With the printer off, and the TYPE command activated, allow the text to scroll up until the required paragraph appears. Stop the scroll with a ^S. press ^P, then ^Q, which will start the scrolling again, this time with a simultaneous print-out. Stop it with ^S again when the paragraph is finished. Knock off the printer with another ^P and carry on the scroll by pressing ^Q, or abandon altogether with a ^C.

Don't forget to disengage the page-by-page display with the command [no page] before a print-out. The stern injunction 'Press Carriage Return to Continue' every few lines rather destroys the dignity of a formal letter, as well as no doubt puzzling the recipient. Don't blame the printer.



#### 4 - The Console Command Processor

If you use TYPE with an afn such as \*.doc, leave out the file specification or if you use options then the transient version of TYPE will be loaded instead of the Built-in version.

For example:-

TYPE filename.typ	Display the ASCII file on the current drive.
TYPE d:filename.typ	Display the ASCII file on the designated drive.
TYPE d:filename.typ [NOPAGE]	Displays the ASCII file on the current drive in a continuous stream rather than a screen at a time.
TYPE *.typ	Type every file of the designated filetype on the current disk
TYPE	Enter the TYPE transient command in interactive mode; the program will respond by prompting for the name of the file to type.

#### 4.6.5 USER -Change user area-

Built in command:

USER

USER n

The USER command allows the user to move to another logical area within the same directory; areas are numbered 0-15. It is actually unnecessary, as the CCP will understand a command consisting of a number followed by a colon as a request to change to the designated user area.

USER n

Change user area to user n

USER

Enter interactive USER mode. The system prompts:-

Enter User £:

n:

Switch to user area n

nd:

Switch to user area n and drive d

## 4.7 The CP/M Plus Transient Utility Commands

There are a host of transient utility commands, some of which are extremely handy, though others are of interest only to the programmer or systems integrator. PIP is probably the most generally used of these commands. PIP is like a copyshop, for making duplicates of files. SHOW is the Hotel Manager, telling you what's where and how much room it takes up, very useful if you're about to start an epic file on an already crowded disk! DIR is the librarian. MAC is the linguist, who translates your pidgin English assembler files into the really incomprehensible machine code that computers lap up. SID is the fibre optics probe that wanders round inside a program, inspecting it for flaws. DUMP lays bare a file's innards. SUBMIT is the dealer, who always does his transactions in job lots, never one at a time. Useful fellow, once you get to know him. Last but not least, HELP is the Citizens Advice Bureau.

All these commands must be stored on disk as '.COM' (command) files which are invoked in the same way as the built-in commands. CP/M comes with the essentials. These CP/M Plus standard transient utilities are :-

System Utilities

COPYSYS	4.7.1	Create one or more new system disks by copying the system tracks
DATE	4.7.2	Set or display the date and time
DEVICE	4.7.3	Alter the i/o device configurations
DIR	4.7.4	See what is on the disk
DUMP	4.7.5	Display a file in ASCII and HEX on the screen or printer
ERASE		Erase files (see Built in command)
GENCOM	4.7.6	Attach an RSX to a '.COM' file
GET	4.7.7	Get console input from a file rather than the keyboard
HELP	4.7.8	Display information on using the CP/M Plus utilities
HEXCOM	4.7.9	Convert a '.HEX' file into a '.COM' file
INITDIR	4.7.10	Converts the disk directory to enable file date stamping
PATCH	4.7.11	Check for Update patches
PIP	4.7.12	Copy, combine, convert, print or transmit files
PUT	4.7.13	Put console output into a disk file
RENAME		Rename files (see Built in command)
SAVE	4.7.14	Save memory image to a file
SET	4.7.15	Set file options
SETDEF	4.7.16	Set system options for program loading and batch commands
SHOW	4.7.17	Display disk and drive statistics
SUBMIT	4.7.18	Execute several commands together as a batch
TYPE		Type files (see Built in command)

Programmers Utilities

ED	4.8.1	Create or alter a text file
LIB	4.8.2	Manage a '.REL' file library
LINK	4.8.3	Link '.REL' files together to make a '.COM' file or an RSX
MAC	4.8.4	Assemble a program in absolute rather than relative code.
RMAC	4.8.5	Assemble a program into a '.REL' module
SID	4.8.6	Debug a program interactively
XREF	4.8.7	Cross-reference a program source.

Each of these transients will be dealt with in more detail:-

#### 4.7.1 COPYSYS -The system copier-

COPYSYS copies the CP/M Plus system from a CP/M Plus system diskette to another diskette. Usually, a CP/M Plus system only requires the system on the bootup disc that is used when the machine is switched on, or reset. In this case, COPYSYS is redundant, as it is easier to do a sector-by-sector copy of the boot disc, so that PROFILE.SUB, and utilities used by PROFILE.SUB can be copied as well. If COPYSYS is used, the new diskette must have the same format as the original system diskette. The system tracks are copied from one disk to the others. Unlike the other CP/M transient commands, COPYSYS is configured for the specific hardware and the user should refer to any specific manufacturers instruction for this command.

**Example :-**

COPYSYS                      Copy system from one drive to other(s).

## 4.7.2 DATE -The Time and Date Utility-

The DATE command allows the user to display or set the date and time of day. It can be set to display the time continuously. The date can be set either by a one-line command or interactively. In systems that lack a true clock with battery back-up, the DATE SET command should be inserted into the PROFILE.SUB file, if date-stamping is implemented.

Displaying the time of day is effected by:

B> date<cr>

but the clock must be set properly when the machine is started from cold.

A> date set<cr>

Enter today's date (MM/DD/YY): 04/29/86<cr>

Enter the time (HH:MM:SS): 09:00:00<cr>

Examples:-

DATE	Displays the current date and time.
DATE C	Displays the date and time continuously.
DATE 08/14/82 10:30:0	Sets the date and time.
DATE SET	Prompts for date and time entries.

## 4.7.3 DEVICE -Device assignment utility-

Device is used to deal with the configuration of physical devices and the way that they relate to logical devices. Each implementation has its unique names for the physical ports, and devices, such as the screen, keyboard, serial ports, or parallel ports. DEVICE displays the current logical device assignments and the physical device names. DEVICE can assign logical devices to peripheral devices attached to the computer. DEVICE also sets the communications protocol and speed of a peripheral device, and displays or sets the current console screen size. Using DEVICE, a wide range of options are made available to the user. For example, it is possible to have a terminal connected to a CP/M Plus computer so that either, or both, can be used to run CP/M. Any number of physical devices can be assigned to a logical device, and any number of logical devices can be connected to a physical device. The List device can be two printers etc.

Device has the following options, which are inserted in square brackets after an assignment expression.-

XON	refers to the XON/XOFF communications protocol.
NOXON	indicates no protocol and the computer sends data to the device whether or not the device is ready to receive it.

Some physical devices (ie serial devices) have configurable baud rates (ie the speed of transmission of the device). If configuring such a device using this utility, the following baud rates are acceptable, in square brackets after the assignment expression:-

50	75	110	134	150
300	600	1200	1800	2400
3600	4800	7200	9600	19200

DEVICE uses the following conventions in describing or assigning devices:-

I = Input O = Output S = Serial X = Xon/Xoff

Examples:-

DEVICE	Displays the physical devices and current assignments of the logical devices in the system and prompts for new assignments
DEVICE CONOUT:=LPT,CRT	Assigns the system console output (CONOUT:) to the printer (LPT) and the screen (CRT).
DEVICE AUXIN:=CRT2 [XON,9600]	Assigns the auxiliary logical input device (AUXIN:) to the physical device CRT2 using protocol XON/XOFF and sets the transmission rate for the device at 9600.
DEVICE CRT	Displays the attributes of the physical device CRT.
DEVICE NAMES	Lists the physical devices with a summary of the device characteristics.

#### 4 - The Console Command Processor

DEVICE VALUES	Displays the current logical device assignments.
DEVICE CON:	Displays the assignment of the logical device CON:
DEVICE LST:=NUL:	Disconnects the list output logical device (LST:).
DEVICE LPT [XON,9600]	Sets the XON/XOFF protocol for the physical device LPT and sets the transmission speed at 9600.
DEVICE CONSOLE [PAGE]	Displays the current console page width in columns and length in lines.
DEVICE CONSOLE [COLUMNS=40 LINES=16]	Sets the screen size to 40 columns and 16 lines.

#### Notes:

In the above examples, the logical device names are followed by a colon. The physical device names are hardware dependent, and may vary between computers, the actual physical devices available are displayed following the DEVICE command.

CP/M Plus supports 5 logical device names:

- 1/ CONIN:
- 2/ CONOUT:
- 3/ AUXIN:
- 4/ AUXOUT:
- 5/ LST:

These logical device names are also known by the following additional logical device names:

Device Name	Logical device
CON:	CONIN: & CONOUT:
CONSOLE:	CONIN: & CONOUT:
KEYBOARD:	CONIN:
AUX:	AUXIN: & AUXOUT:
AUXILIARY:	AUXIN: & AUXOUT:
PRINTER:	LST:



## 4.7.4 DIR (with options) -the extended directory lister-

The DIR command (with options) is an enhanced version of the DIR built-in command and displays files in a variety of ways. Whereas the built-in utility can only list the names of drives on the current user area, the transient version of DIR can search for files on any or all drives, for any or all user numbers. It can display the file size, its time of creation, update or use. It can display disc labels too. One or two letters is sufficient to identify an option. You need not type the right hand square bracket.

For example:- To find the exact amount of storage space taken up by each file. Type [size] in square brackets after the directory and drive:

A> dir b:[size]<cr>

Directory For Drive B: User 0

```
B: COMSYS COM 2k : DATE COM 2k : DEVICE COM 4k
B: DIR COM 4k : DIROPT COM 4k : DUMP COM 2k
B: TRANSTEN 2k : TYPE COM 2k : USER COM 2k
B: XREF COM 2k
```

```
Total Bytes = 26k Total Records = 258 Files Found = 10
Total 1k Blocks = 22 Used/Max Dir Entries For Drive B: 44/ 48
```

Example 2. Some text and other files may have strayed onto the System disk. To isolate quickly all except the '.COM' files, type [exclude] followed by \*.com

A> dir [exclude] \*.com<cr>

Name	Bytes	Recs	Attributes	Prot	Update	Create
ADD65	2k	2 Dir RW		None		03/02/84 00:52
ADD65 BAK	2k	2 Dir RW		None		03/02/84 00:50
CPM3 SYS	36k	276 Dir RW		None		02/24/84 05:36
DIRLBL RSK	2k	12 Dir RW		None		02/15/84 15:10
HELP HLP	62k	488 Dir RW		None		02/15/84 15:29
PROFILE SUB	2k	1 Dir RO		None		11/16/83 20:09
SAMPLE	2k	1 Dir RW		None		03/02/84 00:32

```
Total Bytes = 108k Total Records = 782 Files Found = 7
Total 1k Blocks = 102 Used/Max Dir Entries For Drive A: 38/ 64
```

Some other suggestions here could be useful: one is to call up directories of 'busy' disks from time to time and print them out, attaching the print-out slip to the disk sleeve. It is easy to lose track of what is currently on the 'dustbin disks' that everyone seems to have. Another routine is helpful when you know the name of the file you want, but have no idea on which crowded disk it lurks. To type simply DIR means peering at one crowded directory after another. Better to type DIR followed by the specific filename. At least if it is not there the message comes up quickly:

```
A> dir paydoc.aug [u=all dr=all]<cr>
NO FILE
```

(this command has asked DIR to look on all user areas on all logged-in drives and therefore provides a comprehensive search.)

If it is there, the filename is confirmed:

```
A> dir paydoc.aug<cr>
A:PAYDOC.AUG
```

Best of all, for the really organised, is use of the PUT command to build up a composite file comprising all the disk directories, which gives the operator an e

The available options are as follows:-

[ATT]	Displays the current file attributes.
[DATE]	Displays the date and time stamps of files.
[DIR]	Displays only those files that have the DIR attribute.
[DRIVE=ALL]	Displays the files on all on-line drives.
[DRIVE=(A,B,C,...,P)]	Displays files on all the drives specified.
[DRIVE=d]	Displays files on the drive specified by d.
[EXCLUDE]	Displays files that DO NOT MATCH the files specified in the command line.
[FF]	Sends an initial form feed to the printer device if the printer has been activated by CTRL-P.
[FULL]	Shows the name, size, number of 128-byte records, and attributes of the files. If there is a directory label on the drive, DIR shows the password protection mode and the time stamps. If there is no directory label, DIR displays two file entries on a line, omitting the password and time stamp columns. The display is alphabetically sorted. (See SET for a description of file attributes, directory labels, passwords and protection modes.)
[LENGTH=n]	Displays n lines of printer output before inserting a table heading. n is a number between 5 and 65536.
[MESSAGE]	Displays the names of drives and user numbers DIR is searching.
[NOSORT]	Displays files in the order it finds them on the disk.

[RO]	Displays only the files that have the Read-Only attribute.
[RW]	Displays only the files that are set to Read-Write.
[SIZE]	Displays the filename and size in kilobytes (1024 bytes).
[SYS]	Displays only the files that have the SYS attribute.
[USER=ALL]	Displays all files in all user numbers for the default or specified drive.
[USER=n]	Displays the files in the user number specified by n.
[USER=(0,1,...,15)]	Displays files under the user numbers specified.

Examples of DIR with attributes are:-

DIR d: [FULL]	Displays full set of characteristics for all files in user 0 on the drive specified by d.
DIR d: [DATE]	Lists the files on the drive specified by d and their dates.
DIR d: [RW,SYS]	Displays all files in user 0 on the drive specified by d with Read-Write and System attributes.
DIR [USER=ALL, DRIVE=ALL]	Displays all the files in all user numbers (0-15) in all on-line drives.
DIR [U=ALL, DR=ALL]	Same as above.
DIR [exclude] *.DAT	Lists all the files on the current default drive that do not have a filetype of '.DAT'.
DIR [SIZE] *.PLI *.COM *.ASM	Displays all the files of type PLI, COM, and ASM on the current default drive in size display format.
DIR [drive=all user=all] TESTFILE.BOB	DIR displays the filename TESTFILE.BOB if it is found on any on-line drive in any user number.
DIR [size,rw] d:	DIR lists each Read-Write file that resides on the drive specified by d, with its size in kilobytes. Note that d: is equivalent to d:*..*.

## 4.7.5 DUMP -dumping a file in hexadecimal-

DUMP displays on the console or printer the contents of the specified file in a hexadecimal representation in rows of 16 bytes at a time and also does an ASCII representation of the HEX code on the right of the screen. This utility is useful in examining the contents of file that cannot be represented by their ASCII values (such as '.COM', '.REL' files etc.) or in looking at control characters within text (ASCII) files

It may not immediately seem a giant leap forward to be confronted by a screenful of this:

```
A> dump price1st.aug<cr>
CP/M 3 DUMP - Version 3.0
0000: 54 68 69 F3 20 69 F3 20 74 68 E5 20 6D 61 69 EE  Thi. i. th. mai.
0010: 20 63 68 61 70 74 65 F2 20 6F E5 20 74 68 E5 20  chapte. o. th.
0020: 6D 61 6E 75 61 6C AE 20 49 F4 20 64 65 61 6C F3  man.al. I. deal.
0030: 20 77 69 74 E8 20 61 6C EC 20 74 68 E5 20 27 68  wit. al. th. 'h
0040: 6F 75 73 65 77 6F 72 6B A7 20 8D 0A 70 72 6F 63  curework. ..proc
0050: 65 64 75 72 65 F3 20 6F E5 20 74 69 64 79 69 6E  edure. o. tidyin
```

But there are some occasions when this command may prove useful. One example is in the growing field of connecting computers with typesetters. By and large, ASCII characters generated on a computer emerge as the same ASCII characters on the typesetter's screen after transfer, but some appear rather strangely, and - what is worse - some do not appear at all. If a small portion of the text can be isolated at the computer end, made into a separate file and 'dumped' in hex on the screen, like a biologist doing a biopsy, the hex codes of the disappearing characters can be identified and the situation remedied.

Examples are:-

DUMP filename.typ	Display file from current drive
DUMP d:filename.typ	Display file from designated drive
DUMP d:*.TYP	Display first '.TYP' file from designated drive

## 4.7.6 GENCOM -The RSX utility-

Essentially, the GENCOM utility is for modifying '.COM' files to use '.RSX' additions for such purposes as graphics. The GENCOM command creates a special COM file with attached RSX files. When the CCP loads the '.COM' file, it also loads the RSX to the top of the TPA. The GENCOM command can also restore a previously GENCOMed file to the original COM file by stripping off the header and RSX's. GENCOM can also attach header records to COM files and create dummy files to put RSX extensions onto the BDOS on a more permanent basis.

There are three options to GENCOM:-

[LOADER]	Sets a flag to keep the program loader active.
[NULL]	Indicates that only RSX files are specified. GENCOM creates a dummy '.COM' file for the RSX files. The output COM filename is taken from the filename of the first RSX-filespec.
SCB=(offset,value)	Sets the System Control Block from the program by using the hex values specified by (offset,value).

Examples are:

GENCOM PROGNAME RSX1NAME RSX2NAME	Generates a new COM file PROGNAME.COM with attached RSX's RSX1NAME and RSX2NAME. GENCOM looks at the already-GENCOMed file MYPROG.COM to see if PROG1.RSX and PROG2.RSX are already attached RSX files in the module. If either one is already attached, GENCOM replaces it with the new RSX module. Otherwise, GENCOM appends the specified RSX files to the COM file.
GENCOM RSX1NAME RSX2NAME [NULL]	Creates a '.COM' file RSX1NAME with RSX's RSX1NAME and RSX2NAME.
GENCOM PROGNAME	GENCOM takes PROGNAME.COM, strips off the header and deletes all attached RSX's to restore it to its original '.COM' format.

## 4.7.7 GET -Get input from a file-

GET directs the system to take what is normally entered from the keyboard, from a file for the next system command or user program entered at the console. Console input is taken from a file until the program terminates. When GET is used, it is as if a robot has taken over the keyboard. It is similar to SUBMIT, but cannot manage the parameters used in the latter. If the file is exhausted before program input is terminated, the program looks for subsequent input from the console. If the program terminates before exhausting all its input, the system reverts back to the console for console input.

Get has the following options

[ECHO]	Specifies that input is echoed to the console. This is the default option.
[NO ECHO]	Specifies that file input is not echoed to the console. The program output and the system prompts are not affected by this option and are still echoed to the console.
[SYSTEM]	Specifies that all system input is immediately taken from the disk file specified in the command line. GET takes system and program input from the file until the file is exhausted or until GET reads a GET console command from the file. With the SYSTEM option, the system immediately goes to the specified file for console input. The system reverts to the console for input when it reaches the end of file. Re-direct the system to the console for console input with the GET CONSOLE INPUT FROM CONSOLE command as a command line in the input file.

Examples of GET are:-

GET FILE KEYIN ! MYPROG	Tells the system to activate the GET utility. Since SYSTEM is not specified, the system reads the next input line from the console and executes MYPROG. If MYPROG program requires console input, it is taken from the file KEYIN. When MYPROG terminates, the system reverts back to the console for console input.
GET FILE XIN2 [SYSTEM]	Immediately directs the system to get subsequent console input from file XIN2 because it includes the SYSTEM option. The system reverts back to the console for console input when it reaches the end of file in XIN2. Or XIN2 may redirect the system back to the console if it contains a GET CONSOLE command.

GET CONSOLE

Tells the system to get console input from the console. This command may be used in a file (previously specified in a GET FILE command), which is already being read by the system for console input. It is used to re-direct the console input back to the console before the end-of-file is reached.



## 4.7.8 HELP -Finding out how to use CCP commands-

Typing "HELP" displays a list of topics and provides summarized information for CP/M Plus commands. The HELP transient and the data file must be on disk.

This is a quick and useful guide, whatever one's level of experience, to refresh one's memory with the uses and various permutations of all the important commands. It is a command, so the name and a <cr> is sufficient to start the sequence:

A> help<cr>

Type in the name of your problem and press <cr>. The subject is explained in general and for more detailed guidance the user is invited to type '.subtopics' for the further information available, such as examples or options. Do not type '.subtopics'! That is a generalised label. Type '.examples', or '.options', ie the particular subtopic required.

One or two letters is enough to identify the topics. After HELP displays information for your topic, it displays the special prompt HELP> on your screen, followed by a list of subtopics.

? displays a list of the main topics.  
 . followed by a subtopic name accesses the subtopics.  
 . redisplay what you have just read.  
 <cr> returns to the CP/M Plus system prompt.  
 [NOPAGE] option disables the 24 lines per page console display.  
 any key exits a display and returns to the HELP> prompt.

If you already know the general outline of the subject, but just want to brush up on, say, the various options available, it is possible - and much easier - to 'dial' these up directly. For instance, typing:

B> help erase options<cr>

will remind you whether or not you can erase 'wildcard' files on another drive. You will presumably already know how to erase a simple unambiguous file on the current disk drive.

Before you do this, it is, alas, necessary to help you understand the HELP instructions! The various alternatives are set out in computer shorthand, some of which appear in all their glory, as below:

() surrounds an optional item.  
 | separates alternative items in a command line.  
 ^ indicates the Control Key.  
 [] type square brackets to enclose an option list.  
 () type parens to enclose a range of options within an option list.  
 ... preceding element can be repeated as many times as desired.  
 \* wildcard: replaces all or part of a filename and/or filetype.  
 ? wildcard: replaces any single character  
 in the same position of a filename and/or filetype.

Unfortunately, there is no escaping this kind of shorthand. Computer work does present an array of choices, which is highly useful, but it is daunting when the options are summarised like this, in abstract form.

Use of the symbols \* and ? have already been discussed. To illustrate the others a little more graphically, a hot dog stall run by an off-duty computer man would display the following fare:

Bun (Hot Dog, 2 Hot Dogs,..., 4 Hot Dogs) (onions) (mild relish|hot relish) ([6 mustards])

It's as well to spend a minute puzzling out this illustration - and learn the conventions.

One can either type HELP and enter interactive mode or type one of the following:-

HELP topic                      -displays information about that topic.

HELP topic subtopic           -displays information about that subtopic.

Examples:

HELP                              Enter help program interactively

HELP DATE                        Provide help text on the DATE program

HELP DIR OPTIONS                list the options for DIR

#### 4.7.9 HEXCOM - The converter from '.HEX' to '.COM'--

The HEXCOM Command generates a command file (filetype '.COM') from a '.HEX' file as output from an assembler such as MAC, or a 7-bit communication device. It names the output file to the same filename as the input file but with filetype '.COM'. HEXCOM always looks for a file with filetype '.HEX'. It is equivalent to the transient LOAD in previous versions of CP/M.

Example:

HEXCOM d:prognam	Convert prognam.HEX on drive d to
	prognam.COM

## 4.7.10 INITDIR -The directory initialiser-

The INITDIR Command initializes a disk directory to allow date and time stamping of files on that disk. It must be used before date stamping can be used. INITDIR can also recover time/date directory space. It reformulates the disk directory to be compatible with the extended format that is able to store date and time information.

Once the disk directory has been prepared for date stamping, the directory label must be set to the required type of directory stamping. See SET [CREATE, UPDATE, ACCESS], in this section.

Example:

```
B> initdir a:                Initialise drive A:, the program
                             responds interactively with:-
```

```
INITDIR WILL ACTIVATE TIME-STAMPS FOR SPECIFIED DRIVE.
Do you want to re-format the directory on A: (Y/N)?Y
```

```
B> initdir                  Initialise drive A:, the program
ERROR: Unrecognized drive.   responds interactively with:-
DRIVE: B
```

```
Enter Drive: b:
```

```
INITDIR WILL ACTIVATE TIME STAMPS FOR SPECIFIED DRIVE.
Do you want to re-format the directory on drive: B (Y/N)? y
```

```
Directory already re-formatted.
Do you want to recover time/date directory space (Y/N)? n
```

```
Do you want the existing time stamps cleared (Y/N)? y
```

## 4.7.11 PATCH -the utility patcher-

The PATCH command displays or installs patch number n to the CP/M Plus transient command files. The patch number n must be between 1 and 32 inclusive. When a patch is issued for a specific '.COM', '.PRL', or '.SPR' file, instructions are also provided to update the patch number recorded on the file. Many of the CP/M Plus transient files have been updated including PATCH itself.

Examples:-

PATCH filename 2	Patches the filename.COM system file with patch number 2.
PATCH filename	Display the current patch number for filename.COM
PATCH filename prl	Display the current patch number for filename.PRL

## 4.7.12 PIP -The file interchange utility-

PIP (The Peripheral Interchange Program) is the CP/M transient which implements the basic media conversion operations necessary to load, print, transmit, receive, punch, copy, and combine disk files and to enable peripheral devices to communicate.

The two main purposes of the PIP command are to copy and to combine, but there are so many ways in which little extras can be incorporated that any description is liable to drown in its own qualifications.

PIP can be invoked either by typing PIP and entering PIP command mode or by giving PIP the necessary parameters in the command line for one job. Typing PIP < cmdn > Engages PIP, executes the specified command, and returns to CP/M.

In, for example, copying a file, it is destination first, source second:

```
A> pip m: pip.com a: pip.com <cr>
```

copies PIP itself from drive a: to drive m:

When copying onto the same disk, the same procedures apply, except that the name must be changed in the process, as there cannot be two files of the same name on the same disk:

```
M> pip b: robinson.doc b: travers.doc <cr>
```

This is a common procedure where the Travers document for example has got to be altered and 'Robinsonned' before it is suitable for presenting to said customer.

Typing PIP followed by <cr> engages PIP, prompts the user with '\*', and reads command lines directly from the console. If, therefore, you are doing a number of operations with PIP, stop short after the initial word and press <cr>, instead of typing out the whole command in one line (every time). There follows a prompt in the form of an asterisk:

```
A> pip <cr>
CP/M 3 PIP VERSION 3.0
*
```

Now finish off the command instruction, but this time, instead of returning to the usual A> prompt, it returns to the asterisk so that only the last part of the command need be typed. One can then leave PIP by either typing an empty command line (just a carriage return) or a ^C as the first character of the line. In this example, let us assume that PIP is now in M, and files are being transferred from drive A to drive B:

```
A> m: <cr>
M> pip <cr>
CP/M 3 PIP VERSION 3.0
* b: jones3.ltr a: jones3.ltr <cr>
* b: jones5.ltr a: jones5.ltr <cr>
* b: payroll.sep a: payroll.sep <cr>
* <cr>
M>
```

Files may be sent in a group using ambiguous filenames. These Wildcards

are only allowed on the source file specification. The Destination file specification, if used, cannot contain wildcards. For example:

```
➤ pip b:=a:md?????.*<cr>
```

shifts all files pertaining to the managing director from drive A to B, naming them identically. (It also displays on screen as each file is safely transferred)

```
➤ pip b:dmn????.*=a:md?????.*<cr>
```

will not work, even though he is now the Chairman!

```
➤ pip b:=a:.*<cr>
```

copies all files to B from A.

They can also be joined into one larger new file. for example, three files on drive A, can be joined to become one new master file on drive B:

```
➤ pip b:master=a:file1,file2,file3<cr>
```

or they can be gathered from all over the place:

```
➤ pip m:master=a:file1, b:file2,file3<cr>
```

This gathers 1 from drive A, 2 and 3 from B and transfers it to a master file on M. (Notice the syntax - commas, spaces, and lack of commas and spaces.)

This concatenation assumes the files contain ASCII text terminated with a ^Z, for non ASCII files, the [O] option must be included.

With PIP, the destination does not have to be a disk drive. If a hard copy printout is required, for instance, the file is 'pipped' to the special exit door 'LST:' that leads to the printer, or 'List Device' as printers were originally called before they became commonplace. The main entrance/exit door in a computer, called in fact the Input/Output Port or I/O device for short, is AUXIN: and AUXOUT: [Syntax - notice the colon].

For now, the commonest command involving a device will suffice - the instruction to print a file:

```
➤ pip lst:=b:payroll.jan<cr>
```

It may seem a bit simpler just to press ^P and use the TYPE [no page] command, but PIP is a better way to do it, because of all the refinements that can be incorporated. However, if the file contains TAB's, PIP does not expand the tabs, whilst TYPE does, unless the [Tn] option is added to the PIP command.

There are no less than twenty qualifying instructions, or options, that can be hung onto the main procedures outlined above. They consist of one or two letters only and must appear in square brackets at the end of the command. A useful one when transmitting to another device is [e], by which data is 'echoed to the console', ie scrolls up on the screen as it is being sent across, so one can keep track of where it has got to, or indeed that it is the correct file.

```
➤ pip lst:=b:report.jfs[e]<cr>
```

Most of the others are fairly esoteric, but some can be extremely useful:

A Only copies files altered since the last copying session  
 C Requires prompt before going ahead (as when erasing)  
 E Displays data simultaneously on screen. Must therefore be printable characters  
 F Knocks out form feeds (dot commands and the like)  
 L Changes all upper case characters to lower case  
 O Object file transfer, Use for concatenation of non ASCII files.  
 Pn Sets page length to n lines  
 Sxx Only starts copying when it meets the phrase xx (or whatever). The end of the phrase (or 'string') is denoted by a ^Z.  
 Tn Expands any tab's in the text to the next column of width n characters  
 Qyy Quits printing when it meets the string yy (also rounded off by a ^Z)  
 U Opposite of L - changes lower case to caps.  
 V Verifies that data has been copied correctly.  
 Z Zeroes the parity bit. Avoids odd effects during print-out.

So if you now encounter a command like this:

```
A> pip lst:=b:bloggs.asm [nt8SUBR1:^Z qJMP L3^Z]<cr>
```

you would be able to tell in a flash that this is a simple instruction to print (from drive B) the Bloggs assembler file, numbering each line, inserting tabs every 8th column, translating lower case letters into caps, starting to print when it reaches the string 'SUBR1:' and finishing when it reaches the string 'JMP L3'. Hardly worth taking up your time, really!

It is worth repeating the general principle - that computers are like children, in that they love repetitive routines. Some of the PIP sequences are complex, but if one or two of them are capable of being used frequently, it is worth wrestling with them for an hour or so initially. If you use them often, save repetitive keying by making them into a SUBMIT file.

The general forms of PIP command lines are :-

x:=y:afn	Copy all the files represented by afn from drive y to drive x. 'y' may be omitted, and, if so, the currently logged-in drive is selected.
x:ufn=y:	Copy the file given by ufn from y to x. 'x' may be omitted, and, if so, the currently logged-in drive is selected.
x:ufn=y:ufn	Like the above, but x and/or y may be omitted; the default drive is selected for the omitted drive(s).
destination dev=source dev	Copy to the specified destination device from the specified source device.



As a general rule, PIP follows the convention:

PIP <destination>[options] = <source(s)>[options]

where <destination> is a filename, or a logical or pseudo logical device  
and <source(s)> is a filename, a series of filenames separated by commas, a  
logical or pseudo logical device.

Valid logical destination devices are:-

CON: currently assigned console output - usually the VDU  
LST: current assigned printer  
AUX: to the currently assigned auxiliary output device  
AXO: to the currently assigned auxiliary output device  
OUT: patched output device - see note below

There is one special pseudo destination device:-

PRN: send to the currently assigned printer device with expansion to  
every 8th column, addition of line numbers, and form feeds on every  
60th line.

Valid logical source devices are:-

CON: to the currently assigned console input - usually the keyboard  
AUX: the currently assigned auxiliary input device  
AXI: the currently assigned auxiliary input device  
INP: patched input device - see note below

There are two special pseudo source devices:-

NUL: sends 40 nulls - used when transferring files to a destination  
device which requires a delay to receive the text correctly. For  
example early printers required a delay after a carriage return  
character to allow time for the carriage to move across the page.

EOF: sends an end of file marker - used when transferring files to a  
destination device which requires the end of file character to  
terminate the transfer. For example if PIP is being used to transfer  
text between two computers through the AUX devices, the receiving  
device will require an EOF character to be added to the text  
transferred.

Note: that the transfer from a logical device or pseudodevice source is only  
terminated by the sending of a CTRL-Z character.

The patched input and output devices OUT: and INP: can only be used if the  
PIP.COM file has been specially patched. It is very unlikely that any CP/M Plus  
version of PIP.COM has ever been patched.

Examples using PIP command:-

where d: refers to the destination drive specified in d,  
and s: refers to the source drive specified in s.

PIP	Initiate the Peripheral Interchange Program
CP/M 3 PIP VERSION 3.0	
*d:=s:filename.typ	Copy named file from drive s to drive d
*d:newname.*=s:oldname.typ	Copy and rename file to drive d
*<cr>	exit the PIP program
PIP filename.typ=s:	Copy the file from drive s to the current drive
PIP d:[g3]=s:.*	Copy all files from current user on drive s to drive d on user 3
PIP newname.typ[g1]=oldname.txt	Copy and rename the file to user area 1
PIP d:filename.typ=s:	Copy the file from s to drive d
PIP d:=s:filename.typ	Initiate PIP and copy named file from drive s to d
PIP d:=s:.*	Copy all files from source drive s to drive d
PIP d:=s:filename.*	Copy all files with the specified file name from drive s to drive d
PIP d:=s:*.typ	Copy all files with the specified file type from drive s to drive d
PIP LST:=filename.typ	Print named file on the current list device
PIP AUX:=filename.typ	Send named file to auxiliary output device
PIP CON:=filename.typ	Display named file on the console device
PIP filename.typ=AUX:	Copy data from Auxiliary input device to named file
PIP newname.typ=aname.typ,bname.type,cname.typ	Copy and join together ASCII files
PIP d:newname.typ=s:aname.typ,s:bname.type,s:cname.typ	Copy and join together ASCII files
PIP newname.typ=aname.typ[o],bname.typ[o]	Copy and join together non-ASCII files
PIP LST:=aname.typ,bname.typ	Print files in sequence
PIP PRN:=s:name.typ,s:name.typ	print files in sequence with line numbers
PIP filename.typ=CON:	Write whatever is typed on screen to filename.typ

PIP LST:=CON:                      Write whatever is typed on screen to  
                                 printer (until ^Z is typed)

PIP Filename.typ=INP:[B]           Write input from INP: to file using block  
                                 mode

PIP OUT:=CON:                      Send what is typed on the screen to OUT:

PIP OUT:=s:filename.typ           Send filename.typ on drive s to the OUT:  
                                 device

PIP LST:=S:filename.typ[TiOUZND80]  
                                 Type out the file on the currently assigned  
                                 list device, expanding tabs to 10 spaces,  
                                 converting lower case letters to upper,  
                                 adding line numbers and deleting all chars  
                                 beyond col 80.

#### PIP PARAMETERS

This is the full list of options. They can be combined as in the above example but should be delimited by square brackets.

For example - PIP filename.typ=RDR:[B]

[A] - (Archive)      Copy only files changed since the last copy.  
[C] - (Confirm)      PIP prompts for confirmation before each file copy.  
[Dn] - (Delete)      Delete all characters past column 'n'  
[E] - (Echo)          Echo all copy operations to the console during transfer  
[F] - (Formfeed)      Remove form feeds during transfer  
[Gn] - (Get)          Get file from 'n' user area or Goto user area n  
[H] - (Hex)          Check for proper hex format and report any errors  
[I] - (Ignore)        same as H plus ignores ":00" records in the input '.HEX'  
                         files  
[K] - (Kill)          Kill display of filespecs on console.  
[L] - (Lower)        Change all upper case characters to lower case  
[N] - (Numbers)      Add line numbers with leading zeros suppressed  
[O] - (Object)        Object file transfer; ignores end-of-file character (^Z)  
[P] - (Page)          Insert form feed every 60 lines  
[Pn] - (Page)        Insert form feed every 'n' lines  
[Qstr^Z] - (Quit)     Quit copying after text 'str' is found  
[Sstr^Z] - (Start)    Start copying when text 'str' is found  
[R] - (Read)          Read a '.SYS' file  
[Tn] - (Tabs)        Expand tab space to every 'n' columns  
[U] - (Upper)        Translate all lower case characters to upper case  
[V] - (Verify)        Verify copied data by comparing source and destination  
                         files  
[W] - (Write-over)    Writes over R/O files at destination  
[Z] - (Zero)          Zero parity bit on all the characters in the file

All options except C,G,K,O,R,V and W force an ASCII file transfer, character by character, terminated by a ^Z. (The ^Z represents the combination CTRL-Z key).

The PIP parameters can be combined together to provide quite complex file processing facilities.

The option [Gn] may also be used with the destination file specification.

## 4.7.13 PUT -diverting output to file-

The PUT command is a redirection order. Suppose, for example, you would like all the file directories printed out on a piece of paper, one method would be to call them up, a screenful at a time, and print them out with a succession of Screendump commands, but this would produce rather an ungainly printout. Instead you can redirect the screen display (console output) so that it simultaneously creates a file as well. It ceases to write to disk when the A> prompt reappears.

In this case, assuming the System Disk in drive A and a text disk in B, the sequence would be:

```
A> put console to file b:eldir<cr>
A> dir b:<cr>
A> put console to file b:e2dir<cr>
A> dir b:<cr>
```

etc

The files can then be concatenated, using PIP and edited for a neat printout. The same can be done with the utilities, such as HELP.COM, to enable all the Help messages to be printed out, but a fearful amount of ^[T^E^E and such like must be scooped out first before it is printable.

Printer output can also be redirected, so that characters being sent to the printer go straight to console or are recorded on disk:

```
A> put printer to file b:keyboard.exe<cr>
```

followed by:

```
A> pip lst:=b:crvynce.34!pip lst:=b:crvynce.66!pip lst:=b:crvynce.24<cr>
```

what the PUT Transient actually does is to put console or printer output to a file for the next command entered at the console, until the program terminates. Then console output reverts to the console. Printer output is directed to a file until the program terminates. Then printer output is put back to the printer. PUT with the SYSTEM option directs all subsequent console/printer output to the specified file. This option terminates when you enter the PUT CONSOLE or PUT PRINTER command.

PUT has the following options:-

- |           |  |
|-----------|--|
| [ECHO]    | specifies that output is echoed to the console. This is the default option when you direct console output to a file.   |
| [NO ECHO] | specifies that file output is not echoed to the console. NO ECHO is the default for the PUT PRINTER command.   |
| [FILTER]  | specifies filtering of control characters, which means that control characters are translated to printable characters. For example, an ESCape character is translated to ^[. |

[NO FILTER] means that PUT does not translate control characters. This is the default option.

[SYSTEM] specifies that system output as well as program output is written to the file specified by filespec. Output is written to the file until a subsequent PUT CONSOLE command redirects console output back to the console.

#### Examples

PUT CONSOLE OUTPUT TO FILE XOUT [ECHO]  
Directs console output to file XOUT with the output echoed to the console.

PUT PRINTER OUTPUT TO FILE XOUT ! MYPROG  
Directs the printer output of program MYPROG to file XOUT. The output is not echoed to the printer.

PUT PRINTER OUTPUT TO FILE XOUT2 [ECHO,SYSTEM]  
Directs all printer output to file XOUT2 as well as to the printer (with ECHO option), and the PUT is in effect until you enter a PUT PRINTER OUTPUT TO PRINTER command.

PUT CONSOLE OUTPUT TO CONSOLE Directs console output back to the console.

PUT PRINTER OUTPUT TO PRINTER Directs printer output back to the printer.

#### 4.7.14 SAVE -save memory to file-

Save is equivalent to the built-in transient of earlier releases. SAVE copies the contents of memory to a file. To use SAVE, the SAVE command must be issued before the program which reads a file into memory. Your program exits to the SAVE utility which prompts you for a filespec to which it copies the contents of memory, and the beginning and ending address of the memory to be SAVED.

Typing the command SAVE activates the SAVE utility. When the next transient exits, SAVE intercepts the return to the system and prompts the user for the filespec and the bounds of memory to be SAVED.

Because the CCP is loaded into the base of the CCP, the SAVE utility can no longer be a built-in utility as with previous releases of CP/M and must therefore be loaded as an RSX.

SAVE Ver 3.0

Enter file (type RETURN to exit):filename.typ<cr>

If file filename.typ exists already, the system asks:

Delete filename.typ? Y

Then the system asks for the bounds of memory to be saved:

Beginning hex address: 100<cr>

Ending hex address: 400<cr>

The contents of memory from 100H (Hexadecimal) to 400H is copied to the file filename.typ.

Example of SAVE:-

SAVE	Loads the SAVE RSX into high memory
------	-------------------------------------

## 4.7.15 SET

SET activates and controls the mechanism whereby valuable files can be given a measure of security against accidental or unauthorised tampering. The SET utility sets the way the attributes of files are stored in the directory. It does not affect the contents of the file itself. SET initiates password protection and time stamping of files. It also sets the file and drive attributes Read-Write, Read-Only, DIR and SYS. It lets you label a disk and password-protects the label. To enable time stamping of files, you must first run INITDIR to format the disk directory.

## Examples of SET:-

SET [NAME=THEBOOK]	Labels the disk on the default drive as THEBOOK
SET [PASSWORD=JOYCE]	Assigns the password 'JOYCE' to the disk label.
SET [PASSWORD=<cr>]	Eliminates the existing password.
SET filename.typ [PASSWORD=HELLO]	Sets the file password to 'HELLO'
SET filename.typ [PROTECT=READ]	Makes the password for the file to be required for reading, copying, writing, deleting or renaming the file.
SET filename.typ [PROTECT=WRITE]	The password is required for writing, deleting or renaming the file. You do not need a password to read the file.
SET filename.typ [PROTECT=DELETE]	The password is only required for deleting or renaming the file. You do not need a password to read or modify the file.
SET filename.typ [PROTECT=NONE]	Delete the existing password for this file
SET filename.typ [ro]	Make the file read-only
SET filename.typ [rw]	Allow the file to be altered
SET filename.typ [sys]	Make the file a 'system' file
SET filename.typ [dir]	Reset the 'system' attribute
SET filename.typ [archive=off]	Set the file as 'not backed-up'.
SET filename.typ [archive=on]	Set the file as having been 'backed-up'.
SET filename.typ [F1=on]	Set the user-definable attribute 1 (this can be 1 to 4)
SET [PROTECT=ON]	Turns on password protection for all the files on the disk. Password protection must be on before you can assign passwords to files.

SET [PROTECT=OFF]	Disables password protection for the files on disc.
SET M:*.DOC [PASSWORD=JOYCE, PROTECT=WRITE]	Assigns the password JOYCE to all the '.DOC' files on drive M. Each of these files are also given a WRITE protect mode to prevent unauthorized editing.
SET CC.COM [RO SYS]	Sets CC.COM to Read-Only and SYStem status so that it can be accessed from other user areas and cannot be accidentally deleted.
SET [DEFAULT=JOYCE]	Sets 'JOYCE' as a password if one does not enter a password for a password-protected file.
SET [CREATE=ON]	Turns on CREATE time stamps on the disk in the default or specified drive. To record the creation time of a file, the CREATE option must be turned on before the file is created.
SET [ACCESS=ON]	Turns on ACCESS time stamps on the disk in the default or specified drive. ACCESS and CREATE options are mutually exclusive; only one can be in effect at a time. If you turn on the ACCESS time stamp on a disk that previously had CREATE time stamp, the CREATE time stamp is automatically turned off.
SET [UPDATE=ON]	Turns on UPDATE time stamps on the disk in the default or specified drive. UPDATE time stamps record the time the file was last modified.
SET [CREATE=ON,UPDATE=ON]	Allow datestamping for both file creation and update.
SET [ACCESS=ON,UPDATE=ON]	Allow datestamping for both file access and update.
SET B: [RO]	Sets drive B to Read-Only.
SET B: [RW]	Sets drive B: to allow writing as well as reading

note that SET allows three different types of date stamping.

1. Date and time of creation of the file [create=on]
2. Date and time the file was last accessed [access=on]
3. Date and time the file was last updated [update=on]

Only two types can be set at any one time, however:

- A Date of creation and last update
- B Date of last access and last update



## 4.7.16 SETDEF -Set the disc configuration-

SETDEF allows the alteration of the way that the CP/M system loads programs or operates batch tasks. It allows the user to display or define up to four drives for the program search order, the drive for temporary files, and the file type search order. The SETDEF definitions affect only the loading of programs and/or execution of SUBMIT (SUB) files. SETDEF turns on/off the system Display and Console Page modes. When on, the system displays the location and name of programs loaded or SUBmit files executed, and stops after displaying one full console screen of information. CP/M Plus's system disk can now be any of the drives and can even be a "memory disk" if this is implemented. SETDEF is usually operated from PROFILE.SUB as the user will not want to change this sort of configuration very often.

Examples of use:-

SETDEF	Displays current SETDEF parameters.
SETDEF [DISPLAY]	Turns on the system display mode. Henceforth, the system displays the name and location of programs loaded or submit files executed.
SETDEF [NO DISPLAY]	Turns off the system Display mode.
SETDEF [TEMPORARY=M:]	Sets disk drive M as the drive to be used for temporary files.
SETDEF B:,*	Tells the system to search for a program on drive B, then, if not found, search for it on the default drive.
SETDEF [ORDER=(SUB,COM)]	Instructs the system to search for a SUB file to execute. If no SUB file is found, search for a COM file.

## 4.7.17 SHOW -the provider of system information-

The SHOW utility provides information about the disk drives. It takes on some of the uses of CP/M 2.2's STAT command. The command displays information on various system parameters. these include Access mode and the amount of free disk space, the Disk label, Current user number and number of files for each user number on the disk, the number of free directory entries for the disk and the drive characteristics.

Examples of show are:-

SHOW B:	Show access mode for drive B and amount of space left on drive B.
SHOW [USERS]	Displays the current user number and all the users on the current drive and the corresponding number of files assigned to them.
SHOW [SPACE]	Instructs the system to display access mode and amount of space left on logged-in drives.
SHOW d:[DIR]	Displays the number of free directory entries on drive d.
SHOW [DRIVE]	Displays the drive characteristics of the current drive.
SHOW d:[LABEL]	Displays label information for drive d.

## 4.7.18 SUBMIT -Batch mode Processor-

Instead of taking a sequence of commands from the keyboard, CP/M can take them from a file. SUBMIT takes this further and can actually substitute parameters into a file.

When the same series of commands is carried out time and time again, they can all be bundled together in a file, and SUBMIT sets the whole train in motion. Suppose the task on the first day of every month is to gather together the reports from each of the four branches, print them out for the md, together with the balance sheet, erase the original reports and, as it is a 'first thing' job, load the most used commands into 'drive' 'RAM-disc' for the day.

A simple example might be a pair of instructions to delete all the backup and temporary files in drive B:

With the use of ED, make a short file called CLEANUP.SUB, comprising the two lines

```
era b:*.bak<cr>
era b:*.SSS<cr>
```

Then, putting the disk for treatment in drive B, type:

```
A> submit cleanup<cr>
```

or type:

```
A> cleanup.sub<cr>
```

SUBMIT is a useful ally, with considerable flexibility in its marshalling powers. Literally, suppose the Air Vice Marshal at 11 (Fighter) Group wants up to date information from his airfields of A the aircraft readiness state, R the numbers undergoing 2nd line servicing repairs, S the numbers of airmen on sick parade and C the numbers attending church, for any month, the '.SUB' file HOTPOOP.SUB might consist of:

```
type AS1S2<cr>
type RS1S2<cr>
type SS1S2<cr>
type CS1S2<cr>
```

and the information for Leuchars for May would be elicited by:

```
A> hotpoop leuchars may<cr>
```

where the first variable '\$1' may be substituted by Leuchars, Wattisham, Binbrook, or any of the relevant airfields, and the second variable '\$2' by whichever month is under scrutiny.

As you can see, this arrangement of unloosing a string of commands, each capable of several variables, makes it a powerful device and well worth the initial effort of putting a '.SUB' routine together.

Taking another example, if one wants to do an assembly, followed by a load, followed by an edit of the assembly '.PRN' file to make a new '.ASM' file with the errors corrected, then it is possible to construct the following file with ED

```

mac $1.aaa<cr>
load $1<cr>
era $1.bak<cr>
ren $1.bak=$1.asm<cr>
ren $1.asm=$1.prn<cr>
ed $1.asm<cr>
<UNFABET<cr>
<MIC13D1LOTT<cr>
<B<cr>

```

which takes an '.ASM' file, assembles it, loads it, and unless a key is pressed then makes the old '.ASM' file into a backup file and converts the '.PRN' file into a '.ASM' file with the addition of error messages and leaves the file in ED ready for editing (complete with error codes) and then rerunning the batch.

If the file was called ASSEMBLY.SUB, the batch process would be started with SUBMIT ASSEMBLY filename which would do the batch substituting ufn for \$1 throughout the Batch job.

Should it be necessary to include an actual dollar symbol '\$' within the submit file, type in two dollars '\$\$' and the SUBMIT process will replace each pair of dollars with one dollar.

A '.SUB' file can contain any of the following:-

- 1/ a CCP command line with or without parameters
- 2/ a program command line with or without parameters
- 3/ a comment line
- 4/ a SUBMIT command with or without parameters

A program command line is any entry that is normally typed from the keyboard as input to a program. The program command line must start with a less than symbol '<' as the first character in the line. All remaining characters in the line are used as input as if typed in response to prompts within a program. (This replaces the XSUB utility provided in earlier releases of CP/M).

A comment line starts with a semi colon ';' as the first character in the line. All remaining characters are then ignored.

SUBMIT files can be "nested", that is a CCP command line can contain a SUBMIT command, which on completion will return to the first submit file.

There can be several parameters, and SUBMIT files can do an infinite loop by having, as a last line, a submit to itself.

A special type of submit file is the 'PROFILE.SUB' submit file. If this file exists on the system disk, it is automatically executed whenever the CP/M Plus operating system is first loaded. The PROFILE.SUB file cannot contain parameters. This PROFILE.SUB file is a very useful method of automatically initialising the computer every time the CP/M Plus is loaded. For example, PROFILE.SUB may contain the following commands:

```
; Set drive search path to default, M:, A:, and temporary drive to M:<cr>
SEIDEF *,M,a[temp=m]<cr>
; Set SIO to 9600 baud<cr>
DEVICE SIO [9600]<cr>
; Copy transient files onto drive M:<cr>
pip m:=a:*.com[sys]<cr>
; Request date & time<cr>
date s<cr>
```

Examples of use:-

```
SUBMIT                                initiate interactive mode

SUBMIT filename <param 1> --- <param n>
                                Initiate batch processing using
                                the file filename.SUB and substituting
                                parameters 1--n
```

## 4.8.1 ED -The CP/M text editor-

ED is the CP/M utility with the worst reputation for ease-of-use. It certainly allows for the entry of text in a rather primitive sort of way, but it is very tricky to use when editing or inspecting it. To make a CP/M system at all useful, the first thing to get hold of is a screen-based text-editor. ED was conceived before the widespread availability of good CRT devices (screens). To enter a program or text, one had to use a Teletype machine. This is a sort of electronic typewriter. The user keys in the text, and the result is printed with a great clattering sound on paper. The computer is able to print out on the paper. (In fact, the keyboard presses go to the computer, and the computer controls the printout.) To experience some of the frightfulness of this form of computer interaction, try routing the console output to your printer using DEVICE. This explains the historical reason for BASIC using a PRINT command to place character on the screen.

ED represented the only viable method of entering and editing text under such circumstances. Technology has moved on, but ED has not. However, it is useful if you cannot afford a commercial screen-based text editor, and lack the skills to install a public-domain one.

ED does, however have considerable uses, even if one has an expensive and fancy wordprocessing program. It comes into its own as a means of processing a whole file as part of a batch job.

The user must specify how much (if not all) of the text to be edited must be read from file into the text buffer. When the text is in the buffer, he must then use the ED commands to perform the actual editing. It is best to use line numbers when referring to the location within the text to be modified, as the other method of 'imaginary cursors' or pointers is rather tricky.

ED is invoked by typing ED filename.typ (ufn)

After having invoked ED, then a command such as UV&AB&T (convert input to upper case, use line numbers, put as much as possible of the file into the buffer, and then type it all out.) will get things started.

ED commands tend to be rather obscure and are best remembered as strings to perform specific operations.

ED is a very useful program for particular jobs such as text substitution, column deletion, and the like that would be slow and laborious. If used together with SUBMIT and GET, it is very handy for long repetitive jobs that are best automated.

## ED COMMANDS

nA	Append n lines to buffer (n=0 - use half of buffer)
&A	Put all the file in the buffer
B	Move pointer to beginning of file
-B	Move pointer to end of file
nC	Move pointer forward n characters
-nC	Move pointer backward n characters
nD	Delete n characters forward
-nD	Delete n characters backwards
E	End edit, close file, return to CP/M
nFstr^Z	Find n-th occurrence of the string 'str'

x::yFstr^Z	Find the first occurrence of the string 'str' between lines x&y
H	End edit, move pointer to beginning of file
I	Insert text at pointer until ^Z typed
Istr^Z	Insert string 'str' at pointer
x:I	Enter Insert mode at the start of line x
x:Istring	Insert line consisting of string at line x
x:Istring^Z	Insert string at the beginning of line x
nJastr^Zbstr^Zcstr	Find 'astr', puts 'bstr' after, and erase text up to 'cstr'
K	Kill the current line
nK	Kill n lines starting at pointer
x:K	Kill line x
x::yK	Kill all lines between lines x and y
x:£K	Kill all the lines from line x onwards
nL	Move pointer n lines forwards
-nL	Move pointer n lines backwards
OL	Move pointer to the beginning of the line
nMcommand	Execute string 'command' n times
Mcommand	Execute string 'command' until an error occurs
x::yMcommand	Execute string 'command' repeatedly between lines x & y
nNstring	Global F--finds n'th string or until end of file
O	Abort ED, start over with original file
nP	Display next n pages of 23 lines (n=0 -current page)
-nP	Display previous page and n subsequent ones
Q	Quit without changing input file
R	Read temporary block move file into buffer at pointer
x:R	Read temporary file into buffer at line x
Rfilename	Read filename.LIB into buffer at current pointer
x:Rfilename	Read filename.LIB into buffer at line x
nSoldstr^Znewstr	Substitute string 'newstr' for next n occurrences of string 'oldstr'
nT	Type n lines
-nT	Type the n lines before the pointer
OT	Type from start of line to the pointer
T	Type from the pointer to the end of the line
OTT	Type the entire line
x:T	Type line x
x::yT	Type from line x to line y
£T	Type all lines to the end of what lies in the buffer
U	Change lower case to upper case (next entry)
-U	Disable upper case conversion
V	Enable internal line number generation
-V	Disable internal line number generation
OV	Give free space and size of the buffer
nW	Write n lines to output file (from start of buffer)
nX	Append next n lines to temporary file "X\$SSSSSS.LIB"
OX	Delete the temporary file "X\$SSSSSS.LIB"
nZ	Pause n/2 seconds (2MHz)
n	Move forward n lines and type one line
<CR>	Move forward 1 line and type one line
-	Move backward 1 line and type one line
n:x	Move to n line number and perform "x" command
:mx	Perform command "x" from current line to line m
n:mx	Move to n and perform command "x" to line number m

These commands can be used in combination to provide more sophisticated editing commands. Examples of combined commands are as follows:-

#### 4 - The Console Command Processor

B&T	Type the entire contents of the buffer
-B-T	Type the last line of the buffer
-UV&AB&T	Enable line nos, disable upper case conversion, fill the buffer with the input file and type it all.
BMSolstring^Znustring^ZOTT	From the beginning of the buffer, substitute nustring for all olstring and type each amended line
BMOL5D0TTL	Erase the first five characters of each line throughout the buffer ,displaying the result
x::yXBzLR	Copy lines from between x and y to z (do OX first!)
x::yMX0TTK	Transfer from between x and y into the temporary file typing each line as it is transferred
BM-B-LX0TTK	This pair of commands will completely invert the line R order using the buffer typing each line as it is processed!

The following are the error or status indicators given by ED --

?	You have typed an Unrecognized Command
>	The Memory buffer is full
£	ED Cannot apply the command the number of times specified
O	Cannot open the LIB file in R command (probably not there)
E	Command aborted.
F	File error

The following are the control characters recognized by ED --

^C	Abort and do a System reboot, losing all the edited text.
^E	Physical <CR LF> sequence (not entered in command)
^H	Delete a Character (destructive backspace)
^I	Logical tab to next eighth column
^J	New line (line feed)
^L	Logical <CR LF> in search (F) and substitute (S) strings
^M	New line (carriage return)
^U	Delete a line
^X	Line delete and backspace
^Z	String terminator
Rubout	Character delete and echo the deletion
Break	Discontinue command



## 4.8.2 LIB -The relocatable file manager-

A REL library is a file that contains a collection of object modules. It is good programming practice to construct programs from a series of '.REL' modules that are then linked together into a '.COM' or '.PRL' file. Often, the programmer will wish to store frequently used modules into a library file. that can then be searched at linkage time by LINK for any procedures, functions or subroutines that are called externally by any of the '.REL' files involved in the linkage. The linker can extract any relevant module that it finds within the library. The LIB utility is used to create such libraries, and to append, replace, select or delete modules from an existing library. It will also give information about the contents of library files. LIB creates and maintains library files that contain object modules in Microsoft REL file format. These modules are produced by Digital Research's relocatable macro-assembler program, RMAC, or any other language translator that produces modules in Microsoft '.REL' file format.

LINK can be used to link the object modules contained in a library to other object files. LINK automatically selects from the library only those modules needed by the program being linked, and then forms an executable file with a filetype of COM.

LIB has several options:-

- I The INDEX option creates an indexed library file of type '.IRL'. LINK searches faster on indexed libraries than on non-indexed libraries but such libraries can only be used by Digital Research's LINK utility.
- M The MODULE option displays module names.
- P The PUBLICS option displays module names and the public variables for the new library file.
- D The DUMP option displays the contents of object modules in ASCII form.

LIB Modifiers:-

Modifiers are put in the command line to instruct LIB to delete, replace, or select modules in a library file. Angle brackets enclose the modules to be deleted or replaced. Parentheses enclose the modules to be selected.

```
Delete    <module>
Replace   <module=filename.REL>
```

If module name and filename are the same this shorthand can be used:

```
<filename>
Select    (modFIRST-modLAST,mod1,mod2,...,modN)
```

Here are some examples:-

LIB filename[P]	Displays all modules and publics in filename.REL.
LIB filename[P]=fileone,filetwo	Creates filename.REL from fileone.REL and filetwo.REL and displays all modules and publics in filename.REL.
LIB TEST=TEST1(MOD1,MOD4),TEST2(C1-C4,C6)	Creates a library file TEST.REL from modules in two source files. TEST1.REL contributes MOD1 and MOD4. LIB extracts modules C1, C4, and all the modules located between them, as well as module C6 from TEST2.REL.
LIB fileno2=fileno3<MODA=>	Creates fileno2.REL from fileno3.REL, omitting MODA which is a module in fileno3.REL.
LIB fileno6=fileno5<MODA=fileB.REL>	Creates fileno6.REL from fileno5.REL, fileB.REL replaces MODA.
LIB fileno6=fileno5<THISNAME>	Module THISNAME is in fileno5.REL. When LIB creates fileno6.REL from fileno5.REL the file THISNAME.REL replaces the similarly named module THISNAME.
LIB fileno1[I]=B:fileno2(PLOTS,FIND,SEARCH-DISPLAY)	Creates fileno1.IRL on drive A from the selected modules PLOTS, FIND, and modules SEARCH through the module DISPLAY, in fileno2.REL on drive B.

## 4.8.3 LINK The relocatable file linker-

The output from a relocating assembler has to be linked into one object module, whether it be a '.COM', '.RSX' or '.PRL' file. Whereas the output from an assembler producing absolute code (such as MAC ) produces '.HEX' files that are loaded with HEXCOM, the '.REL' output of a relocating assembler or compiler must be LINKED. LINK combines relocatable object modules such as those produced by RMAC, BASCOM, PROPASCAL and PL/I-80 into a '.COM' file ready for execution. Relocatable files can contain external references and public. Relocatable files can reference modules in library files. LINK can be requested to search the library files and includes the referenced modules in the output file. See the CP/M Plus Programmer's Utilities Guide for a complete description of LINK-80.

## Options

Use LINK option switches to control execution parameters. Link options follow the file specifications and are enclosed within brackets. Multiple switches are separated by commas.

LINK has the following options:-

- A Additional memory;  
reduces buffer space and writes temporary data to disk
- B -BIOS link in banked CP/M Plus system.
  - 1. Aligns data segment on page boundary.
  - 2. Puts length of code segment in header.
  - 3. Defaults to '.SPR' filetype.
- Dhhh -Data origin; sets memory origin for common and data area
- Gn -Go; set start address to label n
- Lhhh -Load; change default load address of module to hhhh. Default 0100H
- Mhhh -Memory size; Define free memory requirements for MP/M modules.
- NL -No listing of symbol table at console
- NR -No symbol table file
- OC -Output '.COM' command file. Default
- OP -Output '.PRL' page relocatable file for execution under MP/M in relocatable segment, and for creation of RSX files.
- OR -Output '.RSP' resident system process file for execution under MP/M
- OS -Output '.SPR' system page relocatable file for execution under MP/M
- Phhhh -Program origin; changes default program origin address to hhhh.  
Default is 0100H.
- Q -Lists symbols with leading question mark
- S -Search preceding file as a library
- \$Cd -Destination of console messages. d can be X (console), Y (printer), or Z (zero output). Default is X.
- \$Id -Source of intermediate files; d is disk drive A-P.  
Default is current drive.
- \$Ld -Source of library files; d is disk drive A-P.  
Default is current drive.
- \$Od -Destination of of object file; d can be Z or disk drive A-P.  
Default is to same drive as first file in the LINK command.
- \$Sd -Destination of symbol file; d can be Y or Z or disk drive A-P.  
Default is to same drive as first file in LINK command.

Examples of LINK commands:-

LINK b:filename[NR]	LINK uses as input filename.REL on drive B and produces the executable machine code file filename.COM on drive B ready for loading at location 0100H. The [NR] option specifies no symbol table file.
LINK file1,file2,file3	LINK combines the separately compiled files file1, file2, and file3, resolves their external references, and produces the executable machine code file file1.COM.
LINK filename=file1,file2,file3	LINK combines the separately compiled files file1, file2, and file3 and produces the executable machine code file filename.COM.
LINK filename,libfile[s]	The [s] option tells LINK to search libfile as a library. LINK combines filename.REL with the referenced subroutines contained in libfile.REL on the default and produces filename.COM.
LINK filename[a,Sim]=file1 & (file2) (file3)	The & indicates a multiple line command. LINK combines the separately compiled files file1, file2, and file3 and produces the executable machine code file filename.COM and the two overlays file2.OVL and file3.OVL. The [a,Sim] option is usually essential with large programs to enable virtual memory space and highest speed using the ram disc for the intermediate virtual files.

## 4.8.4 MAC -the CP/M assembler-

MAC, the CP/M Plus macro assembler, reads assembly language statements from a file of type '.ASM', assembles the statements, and produces three output files with the input filename and filetypes of '.HEX', '.PRN', and '.SYM'. MAC is the CP/M Standard Macro Assembler. The facilities of MAC include assembly of Intel 8080 microcomputer mnemonics with Z80 extensions, along with assembly-time expressions, conditional assembly, page formatting features, and a macro processor which is compatible with the standard Intel definition (MAC implements the mid-1977 revision of Intel's definition). RMAC is similar except it produces '.REL' (relocatable) files. As RMAC cannot produce '.HEX' files that are useful for patching '.COM' files, both are supplied on disc.

The CP/M Assembler is invoked by typing MAC ufn or MAC ufn Sparms. Assembly files must have the filetype '.ASM'. In the case of the first command, MAC creates a '.PRN' (print) file, showing source, error symbols, and a hexadecimal representation of the assembled code with its address. It also creates a '.HEX' file of the assembled code in Intel "HEX" format rather than binary code. Filename.HEX contains INTEL hexadecimal format object code. Filename.PRN contains an annotated source listing that you can print or examine at the console. Filename.SYM contains a sorted list of symbols defined in the program. If options are provided, these files are created according to the parameters specified. Use options to direct the input and output of MAC. Use a letter with the option to indicate the source and destination drives, and console, printer, or zero output. Valid drive names are A thru O. X, P and Z specify console, printer, and zero output, respectively.

\* The general syntax is:- MAC filename {Options}

Only 'filename' is required, and it represents a file named 'filename.ASM'. MACRO Library files may be referenced by the program; these files are named 'filename.LIB'.

MAC filename	Assemble named MAC file on the current drive
MAC d:filename	Assemble named MAC file on the designated drive
MAC SAMPLE SPB AA HB SX	a=source file drive; b=HEX file destination drive (Z=skip); B=PRN file destination drive X=console, Z=skip)

MAC requires approximately 12K of machine code and table space, along with an additional 2.5K of I/O buffer space.

#### Assembly Options That Direct Input/Output

A	source drive for '.ASM' file (A-O)
H	destination drive for '.HEX' file (A-O, Z)
L	source drive for macrolibrary '.LIB' files called by the MACLIB statement.
P	destination drive for '.PRN' file (A-O, X, P, Z)
S	destination drive for '.SYM' file

In the case of the A, H, L, P, and S parameters, they may be followed by the drive name from which to obtain or to which to send data, where --

A,B,C,D -- designates that particular drive  
 P -- designates the LST: device  
 X -- designates the user console (CON:)  
 Z -- designates a null file (no output)

#### Assembly Options That Modify Contents Of Output File

+L lists input lines read from macro library '.LIB' files  
 -L suppresses listing (default)  
  
 +M lists all macro lines as they are processed during assembly  
 -M suppresses all macro lines as they are read during assembly  
 \*M lists only hex generated by macro expansions  
  
 +Q lists all LOCAL symbols in the symbol list  
 -Q suppresses all LOCAL symbols in the symbol list (default)  
  
 +S appends symbol file to print file  
 -S suppresses creation of symbol file  
  
 +1 produces a pass 1 listing for macro debugging in '.PRN' file  
 -1 suppress listing on pass 1 (default)

The programmer can intersperse controls throughout the assembly language source or library files. Interspersed controls are denoted by a "S" in the first column of the input line followed immediately by a parameter.

When the assembler finishes, it will have displayed any error messages and reports any fatal errors. These are as follows:-

NO SOURCE FILE PRESENT	Usually caused by file on wrong drive, the drive being improperly specified, or not having the correct '.MAC' filetype.
NO DIRECTORY SPACE	In creating either the '.HEX' or '.PRN' files, the assembler found the directory full up. Erase some surplus files.
OUTPUT FILE WRITE ERROR	Either the destination disk is write protected or the disk is full up.
CANNOT CLOSE FILE	Your disk is probably write protected.
SOURCE FILE NAME ERROR	There are illegal characters in your filename. This is generally caused by an attempt to use an ambiguous filename.
SOURCE FILE READ ERROR	Generally caused by the source file being mangled
UNBALANCED MACRO LIBRARY	No ENDM encountered for a MACRO definition
INVALID PARAMETER	Invalid assembly parameter was found in the input line.

**MAC error codes**

If the assembler comes across errors in the '.MAC' source it displays them on the console and embedded in the '.PRN' file in the format.

<code> <address> <machine code> label mnemonic operand ;comment.

B Balance error: MACRO or conditional assembly doesn't terminate properly  
 C Comma was not used to delimit items properly.  
 D Data element cannot be placed in data area (It may be too long)  
 E Expression error (ill-formed expression or expression too long)  
 I Invalid character: a non-graphic character has been found  
 L Label error (usually occurs when the label is defined more than once)  
 M MACRO overflow error: internal MACRO expansion table overflow  
 N Not implemented. (possibly an RMAC directive!)  
 O Overflow (expression too complicated or no. labels has exceeded 9999)  
 P Phase error (label defined twice or different values on each pass.)  
 R Register error (specified value not compatible with op code)  
 S Statement/Syntax error: statement is ill-formed  
 U Undefined label (label does not exist ie. has not been defined)  
 V Value error (operand improper, caused often by typing error)

The assembler signs off with the following:-

```
xxxx
yyyH USE FACTOR
END OF ASSEMBLY
```

where xxxx is the Hex address of the program/data end and yyy / OFFH is the fraction of the symbol table space actually used

Assembly source can have line numbers. Labels must start with a letter, ?, or. and can be 16 chars long, though only the first six are significant. They may be followed by a colon. Symbols (eg. EQU) must have no colon.

**Assembly Program Format (spaces or tab separates fields):-**

```
label: opcode operand(s) ;comment
```

Constants can be represented in Binary, Octal, Decimal or Hex radix. The default radix is Decimal: Otherwise, all numbers must be followed by the radix identifiers Q or O (octal), H (Hex), B (binary), or D (decimal). MAC expects all numbers to start with a valid digit (hence 0F000H with leading 0)

**MAC Operators (unsigned 16 bit)**

a+b	a added to b	a-b	difference between a and b
+b	0+b (unary addition)	-b	0-b (unary subtraction)
a*b	a multiplied by b	a/b	a divided by b (integer)
a MOD b	remainder after a/b	NOT b	complement all b-bits
a AND b	logical AND a & b	a OR b	bit-by-bit OR of a & b
a XOR b	logical XOR a & b	a SHL b	shift a left b bits, zero fill
a SHR b	shift a right b bits, zero fill		

## Hierarchy Of Operations in MAC

highest:	*	/	MOD	SHL	SHR	( equal precedence, processed left to right)
	^	-	+			
		NOT				
		AND				
lowest:	OR	XOR				

## Assembler Pseudo-ops

ORG <const>	Set program/data origin (default=0) several can be used.
END <start>	End program (optional). address where execution begins
<label> EQU <const>	Define symbol value(may not be changed )
<label> SET <const>	Define symbol value(may be changed later)
IF <const>	Assemble following block, if <const> is true, until ENDIF.
ENDIF	Terminate conditional assembly block
DS <const>	Define storage space for later use of <const> bytes
DB byte(,byte...,byte)	Define data (bytes) as numeric
DB "<string value>"	Define string data (accepts only printable ASCII chars)
DW word(,word...,word)	Define word(s) (two bytes) as numeric. Low byte=LSB
ELSE	Alternate to IF
ENDM	End of Macro definition
EXITM	Terminate expansion of the current MACRO level
IRP "<string val>"	INLINE MACRO with string substitution
IRPC "<string val>"	INLINE MACRO with character substitution
LOCAL <var1...,varn>	Define LOCAL variables unique to each MACRO repetition
MACLIB <filename>	Specify MACRO Library to load
MACRO	Defines beginning of a MACRO
PAGE <pagelength>	Defines the listing page size for output
REPT	Defines the beginning of a INLINE MACRO
TITLE "<string val>"	Enables page titles and options

A <const> is considered true if the least significant bit is true. It can be a constant or an expression.



#### 4.8.5 RMAC -the relocatable assembler-

RMAC is a version of MAC. It is a relocatable macro assembler that assembles '.ASM' files into .REL files that you can link to create .COM files.

RMAC options are preceded by a \$ sign. They specify the destination of the output files. Output of the SYM or PRN files can, like MAC, be directed to the console (X), the printer (P), or eliminated altogether (Z). The letters A-O can specify drives.

R- refers to the REL file (A-O, Z)

S- refers to the SYM file (A-O, X, P, Z)

P- refers to the PRN file (A-O, X, P, Z)

for example:-

RMAC MYPROG SPX SB RB

Assembles the file MYPROG.ASM from drive A, sends the listing to the console, putting the symbol table on MYPROG.SYM on drive B and puts the relocatable object file (MYPROG.REL) on drive B.

## 4.8.6 SID - The Symbolic Instruction debugger-

The SID Program allows dynamic interactive testing and debugging of programs generated in the CP/M environment. It can be used for altering non-ASCII files, copying files, loading '.HEX' files or examining memory, but is mainly to examine, monitor, test, and alter the way '.COM' files function. It supports real-time breakpoints, fully monitored execution, symbolic disassembly, etc. It is invoked by --

SID	enter SID command mode
SID filename.typ	Enter and place filename.typ into the TPA
SID filename.HEX	Enter and load hex file into binary
SID filename.COM	Enter and place command file for debugging

where 'filename' is the name of the program to be loaded or tested.

SID responds to the normal CP/M input line editing characters.

## SID COMMANDS

<b>A</b> (assemble)	
A <start ad.>	Enter assembler code; start at <start ad.>
<b>C</b> (Call)	
C <start ad.>	Call subroutine at <start ad.>
C <start ad.> <v1>, <v2>	Call <start ad.> with V1 in BC and V2 in DE
<b>D</b> (dump)	
D	Dump RAM from <current ad.>; 16 lines
D <start ad.>	Dump RAM from <start ad.>; 16 lines
D <start ad.>, <end ad.>	Dump RAM from <start ad.> to <end ad.>
<b>E</b> (Employ)	
Efilename	Load <filename> for execution
Efileone, filetwo	Load <fileone> and use <filetwo> for symbols
E*filename	Load filename.SYM for the symbol table
<b>F</b> (fill)	
F <start ad.>, <end ad.>, <const>	Fill RAM from <start ad.> thru <end ad.> with constant
<b>G</b> (Goto)	
G	Start program execution from saved PC
G <start ad.>	Start program execution from <start ad.>
G <start ad.>, bp1	Start at <start ad.> and stop at bp1
G <start ad.>, bp1, bp2 at:	Start at <start ad.> and stop at bp1 or bp2
G, break point 1, break point 2	Start at <current ad.> and stop at bp1 or bp2
(bp denotes a program break point)	
<b>H</b> (Hex)	
H, a, b	Display hex a+b and a-b
<b>I</b> (input)	
I filename	Set up FCB at 005C for R command or program being debugged. (Zap the default FCB)
<b>L</b> (List)	
L	Disassemble from <current ad.>; 12 lines
L <start ad.>	Disassemble from <start ad.>; 12 lines
L <start ad.>, <end ad.>	Disassemble from <start ad.> thru <end ad.>
<b>M</b> (move)	

M <start ad.>,<end ad.>,<new ad.> Move RAM block from <start ad.> to  
<end ad.> to <new ad.>

P (Pass)  
P <breakpoint> Set pass point at address <breakpoint>  
P <breakpoint>,<pass value>  
R (Read)  
R Read file specified by I command to RAM  
R offset Do Read to normal address 0100H + offset

S (Substitute)  
S <start ad.> Substitute into RAM starting at <start ad.>

T (Trace)  
T n Execute n instructions (default=1) with  
register dump (trace)

U (Untrace)  
U n Execute n instructions (default=1) with  
register dump after last instruction

V (Value)  
V Display current values of SID parameters

W (Write)  
W ufn,<start ad.>,<end ad.> Write specified contents of memory to disk

X (Examine)  
Xr Examine/change specified registers or flags where r may be  
C Carry flag Z Zero flag  
M Minus (sign) flag I Interdigit Carry flag  
A Accumulator B BC Reg pair  
D DE Reg pair H HL Reg pair  
S Stack pointer P PC

X Examine all registers

? signifies an error, it can mean:-  
1/ file cannot be opened:  
2/ checksum error in HEX file:  
3/ Assembler/Disassembler overlayed:

SID only understands the op codes for the Intel 8080 microprocessor which is used by all the CP/M Plus programs. But as the Amstrad computers use the more powerful Z80 microprocessor, the Amstrad utilities, for example, include the additional op codes of the Z80 which SID cannot understand. Anyone writing assembler programs for the Amstrad computers is also likely to take advantage of these additional op codes. See section 7.6.1 for a solution.

## 4.8.7 XREF -Cross Reference utility-

XREF is a utility for assembly programmers. It is designed to keep track of the usage of symbols in an assembly language program and record where they occur within the program. This is useful for reference while maintaining a program and also points up routines that, through program modification, are no longer used. XREF also provides a useful cross-reference summary of variable usage in a program. XREF requires the '.PRN' and '.SYM' files produced by MAC or RMAC for input to the program. XREF will also accept '.PRN' files produced by the Microsoft Assembler M80, and the '.SYM' file produced by the linker LINK. The '.SYM' and '.PRN' files must have the same filename as the filename in the XREF command tail. XREF outputs a file of type '.XRF'.

Examples of XREF in use are:-

XREF d:filename	crossreference filename.PRN and filename.SYM to produce a filename.XRF
XREF d:filename SP	crossreference filename.PRN and filename.SYM to the list device.

## CHAPTER 5 – Communications with CP/M

Computers cannot be self-enclosed systems. A computer needs to communicate, if only to a printer to print out a file. The easiest way for two computers to communicate together is for the user to take the disc out of one and put it in another. This is remarkably error-free and effective. Due to the vagaries of the postal system this can be slow, though. It certainly does not help when you are sending a series of characters to a printer. To enable computers to communicate with other devices, such as modems, printers, plotters, acoustic couplers, and other makes of computer, they have plugs and sockets, known as I/O 'ports'. The whole subject of 'Comms' is a grey and frequently misunderstood area, surrounded with quite undeserved mystique, so much so that the beginner is tempted to call in expensive professional help. This is not necessary (nor is it always effective, it must be said), but a certain amount of clear thinking is needed.

### 5.1 Logical and Physical Devices

The first source of confusion is the distinction between logical and physical devices. What are they and which is which?

If you were stopped at a road checkpoint in the morning and asked where you were going, you might reply 'to the office'. You could truthfully give the same reply every day for ten years, even though you might have changed jobs several times in that period. It would still be the correct LOGICAL answer, even though the PHYSICAL location of your office had changed.

In computing, you can likewise give the same logical command 'transmit this file to Output Port A' many times, but you may change the physical device it is being transmitted to on each occasion. The complete list of logical and physical devices recognized by the AMSTRAD micros is as follows:

Logical Device	Physical Device
CONIN: Console input	KEYBOARD
CONOUT: Console output	CRT (Screen)
LST: List output	LPT (Printer, or List Device)
AUXIN: Auxiliary input	SIO (Serial port)
AUXOUT: Auxiliary output	

### 5.2 Physical Devices

First, we shall consider physical devices, as they are easier to visualize. Like the description of hardware in the introduction, you can kick them! Logical 'devices' sound equally physical, but are better thought of as more like bus routes. Physical devices themselves fall into one of two categories, depending on whether they are fed through a serial or a parallel port.

### 5.2.1 Serial and Parallel

Data can be transmitted either in parallel or serial form. Transmission in parallel is how greyhounds emerge from the traps - all side by side and simultaneously. Serial transmission is a nose-to-tail affair, like elephants at the circus. It requires only two wires (or at most three, when data is being acknowledged), but is slower. Though parallel transmission is obviously faster, it is only good over short distances. There is one parallel built-in to the machine to deal with the printer, and two optional serial ports for general purpose use for the Amstrad 6128 and PCW 8256. The only point to bear in mind here is that a serial device cannot be connected to a parallel port, or vice versa. Apart from that, the difference is academic for our purposes and we shall not distinguish between them, apart from tagging them (S) or (P).

### 5.2.2 The Printer (List Device) (P) or (S)

The Printer needs little comment, except for a mental reminder that it is often referred to as the List Device (best remembered by the term 'listing paper').

NB: because printers normally sit next to the computer that drives them, most are constructed to receive the faster, parallel type of transmission, but this is not always so. There are serial printers on the market, in which case a certain amount of re-routing has to be done (see logical assignments below).

### 5.2.3 Plotters (P)

Plotters are a specialist type of printer used for graphics work though most modern ones can be persuaded to write text as well, and thereby act as a printer for short runs. (The pens soon run out!)

### 5.2.4 SIO (S)

The main serial 'device' is named SIO on the Amstrad machines, though it is likely to be different on other machines. It is an RS232C port. We will deal with the nuts and bolts of this interface and how connections with it are made. Assume for the time being that it is easy; (a rather optimistic assumption)

The Serial interface SIO, which is an optional extra on the Amstrad 6128 and PCW 8256 in fact is the conventional thin D shape and for convenience sake this type of physical device is referred to as 'RS232', but we should remember that this is a slight misuse of language.

The SIO is the main serial port to the outside world, and because most communications are serial, it could be thought of as the most important 'device'. As such it is capable of a good deal of re-rigging, or 're-configuration', in order to match other devices that are less flexible than the Amstrad 6128 and PCW 8256.

The user has full control over which logical device the SIO is assigned to (normally AUXIN: or AUXOUT:, but it could be the console CON: or the LST:), and also the speed and type of transmission. The socket and the wiring configuration to the socket are unalterable by keyboarding, but if anything other than a 'straight through' wiring connection (pin 1 to pin 1, 2 to 2, etc)

is required by the other device - which it frequently is - then any kind of Clapham Junction arrangement can quickly be achieved by a soldering iron and a short length of connecting cable.

#### 5.2.4.1 Options available with SIO

As has been explained, an SIO is not really the physical device that the user has in mind when he wants to connect with other equipment. It is merely a socket which other equipment may or may not successfully be plugged into. So what will work? These, for a start:

1. Other computers                      Any computer with an RS232 I/O port can serve for interchange of data. It helps if the data is 'ASCII' (American Standard for Communication Information Interchange).
2. Teletype terminals
3. Modems                                An intermediate 'black box', connected at one end to the Amstrad 6128 and PCW 8256 and at the other direct to the telephone socket, MODulates the computer output to speech frequency, so that it can be transmitted down a normal telephone line, then DEModulates it the other end to the frequency acceptable to the receiving device. A variety of and modes of transmission is possible.
4. Acoustic couplers                   These are inexpensive modems, which are fitted with a pair of rubber cups to accommodate the telephone handset, so that the telephone wiring need not be tampered with. They are restricted to a rather slow speed, but they are much more portable.
5. Serial printers
6. Typesetters                         In computer language, these are ultra high resolution printers, that achieve their quality by photographic or laser, rather than mechanical, means.
7. Other                                 A huge array of specialist systems can be driven by micro computers, such as thermostats, watches, robots, security systems, digitizers, telemetry devices and things that open garage doors. Whatever the gadget, all the 'driving' is done via the RS232 serial port.

Some comments on actually making communication with these devices appear below. Meanwhile let us not forget the ports leading to other parts of the computer, such as the keyboard and screen. They may seem 'internal doors' by comparison to the 'outside doors' that communicate via an acoustic coupler to another computer hundreds of miles away, but as far as the computer is

concerned they are still doors, or ports, to another device.

### 5.2.5 The Keyboard and Screen, or CRT

There is no need to describe these as physical devices, but their input and output can be rerouted.

## 5.3 Logical Devices

As mentioned above, these are best thought of like bus routes, which can be switched or reassigned, rather than devices as such. For example, when a file is printed out, by whatever command, it is sent down the logical route LST: (list device). Normally, this is routed to the parallel port, where a parallel printer will be waiting to print whatever comes down the line. If a serial printer is acquired instead, the command will still be directed to LST:, but the route will be reassigned to an RS232 (serial) port called the SIO on the AMSTRAD.

### 5.3.1 Default Assignments

The default logical assignments are these:

Logical Device		Physical Device
CONIN:	Console input	Keyboard
CONOUT:	Console output	CRT (Screen)
LST:	List output	LPT
AUXIN:	Auxiliary input	SIO
AUXOUT:	Auxiliary output	SIO

### 5.3.2 Restrictions

Many other reassignments are possible, and useful, but not all permutations are allowed. (one cannot get input from the printer!). If you were to assign CON: (the screen and keyboard) to SIO then the machine could be worked from a terminal plugged-in to the serial port. If you assigned CON: to both the SIO and CRT (the keyboard/screen combination on the Amstrad machine) then you could work the machine from both it and the terminal.

## 5.4 Reconfiguration.

There is more to rerouting than just matching up a logical device with a new physical device when dealing with a serial port. Transmission must also be:

1. At the right speed, of 'baud rate' and
2. Performed in the correct manner, or 'protocol'.
3. With the correct type of serial transmission.



#### 5.4.1 Baud Rate

The speed, or baud rate, must match that of the device it is being connected with, and the type of transmission procedure, the protocol, should also match, or the computer should be tricked into thinking it matches.

#### 5.4.2 Protocol

For reliable, error-free transmission, data is handled very positively. Data must also either be asked for, acknowledged, and then asked for again in an elaborate procedure known as handshaking. (This now-send-now-stop-sending is known as 'X-on X-off' and must be mimicked by the other device.) Or else it must be disconnected, changing it to the 'No X-On' mode (which must likewise be mirrored). It is the difference between operating a chain of canal lock gates correctly, and alternatively opening all gates wide and letting barges surfride through, chancing their luck. In practice, 'No X-on' is less hazardous than that, but expert advice is very advisable, as many hours can be wasted, even by experts, trying to fathom out the right cable configuration.

#### 5.4.3 Transmission Type

The receiving device must know how the bytes being transmitted are made up - how many data (start) bits, how many stop bits, and whether the 'parity' is odd, even or nil. A typical protocol is 8 data bits, one stop bit and no parity.

#### 5.4.4 Configuring the Amstrad 6128 and PCW 8256

There is one Amstrad and one Digital Research utility that enable configuration of the serial ports.

1. SETSIO.COM is the Amstrad utility that can set all the serial port options.
2. DEVICE.COM is the utility supplied by Digital Research. It deals with logical-to-physical mapping and can deal with both baud rate and X-on/X-off protocol. It cannot deal with the number of data bits, start bits or stop bits.

#### 5.5 The RS232 INTERFACE

The RS-232 interface is surrounded with a quite unnecessary mystique. It is a bit of a donkey's breakfast because it is being used for purposes for which it was never designed. The main joy of a serial interface, besides its ability to be transmitted for long distances, is that it requires only two wires. Simplicity in the world of microcomputers is difficult to achieve and the full RS232 interface looks more like Bluebeard's chin than a serial interface. Essentially, the RS-232-C interface has two types of signals: these are referred to as control signals and data signals. The main control signals are DTR, RTS, CTS, DSR, and CD. The data signals are TxD and RxD. The voltage levels of these signals are as follows.

- (1) Control signals  
ON: ~5V to ~15V  
OFF: -5V to -15V
- (2) Data signals  
Mark (logical 1): -5V to -15V  
Space (logical 0): ~5V to ~15V

The RS in RS-232-C stands for 'Recommended Standard', 'C' merely denotes that it is the third version of the standard.

The connector is a 25-pin (DB-25) connector. Although there are twenty five pin connections, most of these are unused. Ten pins are the maximum that are customarily used. these are:-

- 1.....protective ground (AA)
- 2.....transmitted data (TxD)
- 3.....received data (RxD)
- 4.....request to send (RTS)
- 5.....clear to send (CTS)
- 6.....data set ready (DSR)
- 7.....signal ground (AB)
- 8.....carrier detect (DCD)
- 20.....data terminal ready (DTR)
- 22.....ring indicator(CE)

Pins 15, 17 and 24 are used only with high speed modems.

The meanings of the various signals are as follows.

#### Protective ground (AA)

This terminal is usually connected to the microcomputer chassis; ordinarily, it is also connected via the external cable to the corresponding terminal on the other device.

#### Transmitted data (TxD)

TxD is the signal used when transmitting data from the microcomputer to the device (acoustic coupler, etc.) with which the Microcomputer is connected. This is possible when the 'Clear to send' (CTS) signal is on.

#### Receive data (RxD)

RxD is the data signal from the acoustic coupler or other RS-232C compatible device to the Microcomputer.

#### Request to send (RTS)

RTS is the signal which controls the communication function of the device (acoustic coupler, etc.) connected to the Microcomputer. The connected device becomes ready to send when this signal is ON.

#### Clear to send (CTS)

CTS is the signal which indicates whether the connected device is ready to accept data transmissions. Transmission is enabled when this signal is ON and disabled when it is OFF.

#### Data Set ready (DSR)

DSR is the signal which indicates whether the connected device is ready for operation. When this signal is ON, the applicable device is connected to the interface cable and is ready to accept data transmission/reception control

signals.

**Signal ground (AB)**

The SG terminal provides an electrical reference potential which is common to all signal lines. It is connected to the corresponding terminal on the connected device.

**Carrier detect (DCD)**

The DCD terminal is used for detecting the carrier signal from the connected device.

**Data terminal ready. (DTR)**

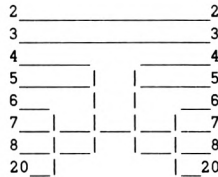
DTR is the signal output by the Microcomputer to the connected device to indicate that it is ready to receive data.

**Ring Indicator. (CE)**

This is a line that is specifically used when connected to a modem and is used by the modem to signal that there is a ring signal on the line. It is only used on equipment that answers incoming calls.

Although only two pins are needed for a one-way communication, it is customary to use the control signals to 'handshake' the data across so that there is some way for the device receiving the data to cry 'enough'. There are two ways for a device to wire the RS-232 device, either as a DCE (data communications equipment) or as a DTE (data terminal equipment). The RS232 pins are named from the point of view of the DTE, so it is the DTE that transmits on 2 and receives on 3. The DCE is the self-effacing one, receiving on 2 and transmitting on 3, though its way of working is not mentioned in the above list. The DTE is the boss. What it does mean, though, is that when making a connection then pins 1 to 7, and 20, wired straight across to their opposite numbers, might well work. A DCE should have a female connector and a DTE should have a male connector; but this standard is generally ignored and enormous confusion has resulted. For communications between similar devices, it is a general rule that, instead of a pin-to-pin connection, the following pins should be reversed in their connections- TxD and RxD, RTS and CTS, DSR and DTR. In other words, instead of connecting pins 1 to 1, 2 to 2, 3 to 3, 4 to 4, 5 to 5, 6 to 6, 7 to 7, 8 to 8 and 20 to 20, some pins should be reversed so that the connections are 1 to 1, 2 to 3, 3 to 2, 4 to 5, 5 to 4, 6 to 20, 20 to 6, 7 to 7, and 8 to 8.

If this does not work, then the chances are that the computer requires 'hardware handshaking' and the DCE does not, or maybe the reverse. In which case one has to fool the computer into thinking it is getting the signals it wants. When the computer is ready to transmit it sends out a signal on DTR (Data Terminal [computer] Ready). It seeks a reply down the DSR (Data Set [modem] Ready) line. If the two are shorted across, it gets it and is happy, not realizing it is its own signal coming back. So pins 6 and 20 are connected, on each side, and for similar reasons, 4, 5 and 8 are connected, in this manner:



More trouble can arise when the computer (which is DTE) is connected to another micro, which does not know if its interface is wired as DTE or DCE and there is nothing in the documentation to say. And what's a printer - DCE or DTE? One can quickly be landed in a massive identity crisis, not to mention a good deal of heavy anthropomorphic repartee ("Make it think it's a modem"). Normally all computers (micros, mainframes or terminals) are DTE and modems are DCE. Another rule of thumb is that a male plug denotes DTE and DCE is female, but this not infallible. Also printers are generally DTE.

they other computers, or devices, it is useful to have a good serial cable that links all ten major signals, and preferably all 25 (so that you know exactly where you stand.) You should have a null modem (two D25 connectors connected back to back with connections as described in the last paragraph) and a 'breakout box' or other gadget that enables you to fiddle with the connections. I prefer to have a whole lot of D25 connectors connected back to back with the connections in the following list already done, so that I can stick them on the end of the cable one after another and thereby save a great deal of time if one of them works.

To summarize:

Find out:

1. Whether the other device is wired as DTE or DCE
2. Which handshaking lines (if any) are used
3. Whether software handshaking is used

Also, for good measure, equip yourself with a breakout box.

If you manage to make a cable that works, guard it with your life.

Here are some of the 80 known ways of connecting with a peripheral via a serial port:-

Epson MX-80 and IDS prism printers		TI, Brother, Epson MX 100, and OKI data printers	
Micro	Printer	Micro	Printer
1 (Prot) <----->	1 (Prot)	1 (Prot) <----->	1 (Prot)
2 (XMT) ----->	3 (RCV)	2 (XMT) ----->	3 (RCV)
3 (RCV) <-----	2 (XMT)	3 (RCV) <-----	2 (XMT)
4 (RTS) ----->	5 (CTS)	4 (RTS) ----->	5 (CTS)
5 (CTS) <-----	20 (DTR)	5 (CTS) <-----	11 (STF)
6&8&20		6&8 <-----	20 (DTR)
7 (Com) <----->	7 (Com)	7 (Com) <----->	7 (Com)
		20 (DTR) ----->	6&8

## 5 - Communications with CP/M

NEC and Anadex printers (note the use of pin 19)	
Micro	Printer
1 (Prot) <----->	1 (Prot)
2 (XMT) ----->	3 (RCV)
3 (RCV) <-----	2 (XMT)
4 (RTS) ----->	5 (CTS)
5 (CTS) <-----	19 (rts)
6&8 <-----	20 (DTR)
7 (Com) <----->	7 (Com)
20 (DTR) ----->	6&8

(note the originality of pin 19,  
the secondary RTS!)

Gume Sprint printers	
Micro	Printer
1 (Prot) <----->	1 (Prot)
2 (XMT) ----->	3 (RCV)
3 (RCV) <-----	2 (XMT)
4 (RTS) ----->	5 (CTS)
5 (CTS) <-----	20 (DTR)
6&8 <-----	4 (RTS)
7 (Com) <----->	7 (Com)
20 (DTR) ----->	6&8

Diablo 620	
Micro	Printer
1 (Prot) <----->	1 (Prot)
2 (XMT) ----->	3 (RCV)
3 (RCV) <-----	2 (XMT)
5 (CTS) <-----	4 (RTS)
6&8 <-----	20 (DTR)
7 (Com) <----->	7 (Com)
20 (DTR) ----->	6 (DSR)

Diablo 630 (Why pin 11, one wonders.)	
Micro	Printer
1 (Prot) <----->	1 (Prot)
2 (XMT) ----->	3 (RCV)
3 (RCV) <-----	2 (XMT)
5 (CTS) <-----	11 (STF)
7 (Com) <----->	7 (Com)
6&8&20 <----->	6&4

Smith-Corona TP-1	
Micro	Printer
1 (Prot) <----->	1 (Prot)
2 (XMT) ----->	3 (RCV)
3 (RCV) <-----	2 (XMT)
4 (RTS) ----->	5 (CTS)
5 (CTS) <-----	4 (RTS)
6&8&20	
7 (Com) <----->	7 (Com)

Cifer Terminal	
Micro	Terminal
1 (Prot) <----->	1 (Prot)
2 (XMT) ----->	2 (XMT)
3 (RCV) <-----	3 (RCV)
4 (RTS) ----->	4 (RTS)
5&12 <----->	5&12
6 (DSR) <-----	6 (DSR)
7 (Com) <----->	7 (Com)
20 (DTR) ----->	20 (DTR)

(This also seem to use secondary DCD  
to keep you on your toes).

Hewlett-Packard Plotters		C-Itoh 4800 Plotter	
Micro	Plotter	Micro	Plotter
1 (Prot) <----->	1 (Prot)	1 (Prot) <----->	1 (Prot)
2 (XMT) ----->	3 (RCV)	2 (XMT) ----->	3 (RCV)
3 (RCV) <----->	2 (XMT)	5 (CTS) <----->	20 (DTR)
5 (CTS) <----->	20 (DTR)	20 (DTR) ----->	5 (CTS)
6&8&20		7 (Com) <----->	7 (Com)
7 (Com) <----->	7 (Com)		

Try using one of these 'null modem' connection to an unknown peripheral:-

Null Modem (Hardware Handshaking)		Null Modem (Hardware Handshaking)	
Your Micro	His Device	Your Micro	His device
1 (Prot) <----->	1 (Prot)	1 (Prot) <----->	1 (Prot)
2 (XMT) ----->	3 (RCV)	2 (XMT) ----->	3 (RCV)
3 (RCV) <----->	2 (XMT)	3 (RCV) <----->	2 (XMT)
4 (RTS) ----->	5 (CTS)	4&5	4&5
5 (CTS) <----->	4 (RTS)	6&8 <----->	20 (DTR)
6 (DSR) <----->	20 (DTR)	7 (Com) <----->	7 (Com)
7 (Com) <----->	7 (Com)	20 (DTR) ----->	6&8
8 (DCD) <----->	8 (DCD)		
20 (DTR) ----->	6 (DSR)		

If this fails, try connecting 5&20.

4 Wire Null Modem (Software Handshaking)		3 wire Null Modem (Software Handshaking)	
Your Micro	His device	Your Micro	His device
1 (Prot) <----->	1 (Prot)	2 (XMT) ----->	3 (RCV)
2 (XMT) ----->	3 (RCV)	3 (RCV) <----->	2 (XMT)
3 (RCV) <----->	2 (XMT)	4&5&20	4&5&20
4&5	4&5	6&8	6&8
6&8&20	6&8&20	7 (Com) <----->	7 (Com)
7 (Com) <----->	7 (Com)		

Occasionally, as above, you will find a device that requires a totally potty signal on a pin that is completely irrelevant. (eg 11, 12, or 19). You really must grit your teeth and read the manual in these cases. Breakout Boxes are useful devices and there are many fancy devices for those of you with too much money. All that is really necessary is a breadboard with a male D25 connected at one end and a female at the other. Once you have a connection that works, then note it down in a notebook. If it differs from any of the ones above or you find out a new one please let us know. Remember that, if you ignore or mislead the control signals, you run the risk of losing data, particularly where there is no software handshaking (Xon-Xoff, Ack-Nak etc). On a printer or plotter, the result of ignoring or misconnecting the control lines is often buffer overflow. This shows itself as loss of lines of text and production of random nonsense. Some interface problems are extremely subtle, and only show up at higher data rates. If you believe that you have cracked an interface problem, then test the line at the highest allowed data transfer rate (baud rate) to be absolutely sure.

This is a tabular description of the RS232 (V24) interface.

Pin	Description	NAME	V24	RS232	DIN	NOM
1	Protective Ground	<-> Prot	101	AA	E1	TP
2	Transmitted Data	--> XMT	103	BA	D1	ED
3	Received Data	<-- RCV	104	BB	D2	RD
4	Request to Send	--> RTS	105	CA	S2	DPE
5	Clear to Send	<-- CTS	106	CB	M2	PAE
6	Data Set Ready	<-- DSR	107	CC	M1	PDP
7	Ground	<-> Com	102	AB	E2	TS
8	Data Carrier Detect	<-- DCD	109	CF	M5	DP
9	Positive Test Voltage	<-- -	-	-	-	9
10	Negative Test Voltage	<-- -	-	-	-	10
11	Select XMT Frequency	--> STF	126	CG	S5	SFE
12	Secondary DCD	<-- dcd	122	SCF	HM5	DPS
13	Secondary CTS	<-- cts	121	SCB	HM2	PAES
14	Secondary XMT	--> xmt	118	SBA	HM1	EDS
15	Transmit Clock	--> Xclk	114	DB	T2	HE
16	Secondary RCV	<-- rcv	119	SBB	HD2	RDS
17	Received Clock	<-- Rclk	115	DD	T4	HR
18	Local Loopback	--> -	141	-	PS3	CB
19	Secondary RTS	--> rts	120	SCA	HS2	DPES
20	Data Terminal Ready	--> DTR	108	CD	S1	CPD
21	Signal Quality	<-- SQL	110	CG	PS2	
22	Ring Indicator	<-- RI	125	CE	M3	IA
23	Data Rate Select	--> DRS	111	CH	S4	SDB
24	External Transmit Clock	--> -	113	DA	T1	HEE
25	Busy - Standby	<-- BY	142	-	PM1	IT

So we have made a connection and made the correct logical assignments, what next?

Lets take an example: You have your nice Amstrad Microcomputer, next to it is a nice eight-inch-disk machine positively groaning with public-domain software. You have a serial interface with a breadboard in the middle to enable you to fiddle with the RS 232 lines. PIP is the vital program to have on your machine. Luckily, the Amstrad machines have the AUXOUT: and AUXIN: (PUN: and RDR: on the 464/664 machines) devices properly implemented for the serial port as a configurable RS 232 device. If so, then match the baud rate, parity, start/stop bits, and hardware protocol to the other computer and type

A> pip con:=aux:<cr>

It should hang there, blinking its cursor vacuously at you. On the other machine, type

A> pip pun:=con:<cr>

The other machine uses CP/M 2.2

and then type in an endearing message on the keyboard. If the endearing message appears on the Amstrad screen, then you have communications. Real life is not, however, like this, and you may need to try another method of connecting the ports together, using some of the techniques we have already described (eg switching lines, looping control lines back, pulling some of the control lines to deck, or coupling control lines together to get success. Hardware engineers unleash all their pent-up creative instincts on the RS 232 pinouts (one manufacturer has only three pins connected). I generally find that I have to do a bit of massaging to get even a simple interface to work.

On a bad day, you will find that the other micro does not have an implemented PUN: or RDR: device. (The Amstrad CPC 6128 and PCW 8256 use the AUXIN: AUXOUT: channel) This requires you to patch in your own driver into PIP to become the INP: OUT: device. The INP: OUT: device is a useful feature in PIP that extends its usefulness enormously. All that is necessary is to write a patch (using CP/M-80 in this example) using ASM that has its ORG (program origin) at 0103H. It will start something like this

```

org      0103H

jmp      GEIBYT
jmp      FUIBYT

Data:    DB      000

FUIBYT:      ;write character in the C register to the port
..
..
..

GEIBYT:      ;fetch next character from the port and place it in DATA
              ;with its high (parity) bit stripped. Hang on the port
              ;until a character is received
..
..
..

;You have a generous patch area of 246 bytes for your serial port
;driver. If the device requires initialisation, then include this the
;first time either routine is called by PIP.
```

You can then assemble this into a '.HEX' file which can be patched into your copy of PIP using SID or DDT. Some computer manufacturers have this patch already done, which is useful. You then substitute INP: for RDR: and OUT: for PUN: in the PIP commands.

PIP is excellent for passing text or '.HEX' files but is tricky to use for transmitting binary files. It will check '.HEX' files for validity and contains routines for a very wide range of needs. People who feel it should be called



COPY.COM probably do not appreciate the facilities in the program. For example, one can use it to transmit files, receive files, write text directly to the printer, concatenate files, filter form feeds from a file, change text files from upper to lower case or the reverse, strip the high bit from ASCII text (for such reasons as converting Wordstar files), eliminating Tab characters, truncate lines in text, number text lines, or paginate text files. It will selectively copy files that have been altered, or ask the operator which files from a wildcard specification should be copied (version 3).

When initially using PIP to transfer files between micros, it is useful to 'echo' the transfer to the screen so that one can see by inspection whether transfer is succeeding. If you are losing characters, the answer could be to use the hardware handshaking or use a software protocol such as X-on X-off. Once you have a comms program on your micro, then this will cease to be a problem as they all have, by necessity, software handshaking and error-detection/recovery within the program itself. I can remember, when desperate, using the 'echo' facility within PIP to slow down the inter-character gap on the transmitting computer in order to cure character loss whilst using PIP to transmit a comms program to one particular micro. (in this case, only use echo on the transmitting computer).

There are a large number of programs and utilities that are written to enable the user to communicate down a telephone line. Obviously, the first thing to obtain is the necessary hardware; an acoustic coupler or modem. An acoustic coupler is a device that makes the whole process of connecting to the telephone line remarkably easy. an acoustic coupler actually 'warbles' through the telephone speaker, and listens for the 'warble' at the other end. The 'warbling' sound and the whistle of the 'carrier' are both enshrined in the standards of the CCITT (in much of Europe) and BELL (in the States and elsewhere) so that acoustic couplers can talk to other acoustic couplers or modems following the same standard. All one has to do in the case of an acoustic coupler is to set the modem to 'originate' mode, dial the number and, when the modem or acoustic coupler at the other end responds with the answering carrier frequency, you switch on the coupler, apply the telephone handset to the coupler, run your communication program, and sit back with a smug grin of satisfaction on your face. When receiving a transmission, answer the call, switch the modem to 'answer' mode, switch the 'carrier' on, place the receiver into the coupler, run your telecommunications program, and, likewise, sit back watching all that information streaming into the computer. With modems, the whole process is a great deal simpler. A simple modem will work in the same way as a coupler with the advantage that you do not have to put the receiver anywhere, and the transmission is of a better quality, but has the disadvantage of lack of portability. A good 'autodial-autoanswer' modem will dial up another modem, perform all the niceties, and hang up all without human intervention. It will automatically answer calls, and if the right program is in place, it will save the data on disc. Real life is, of course, never quite that simple. Once a modem and software is properly installed and set-up, then it will give few problems. The subject of Telecommunications is fundamentally simple, but made maddeningly complex by the needs for faster transmission rate, data integrity, encryption, networking, and so on. Simple asynchronous communications at a pedestrian rate enshrined in CCITT V21 is simple and relatively trouble-free, whereas the more esoteric synchronous systems such as X25 are for medium to large business use only. The average bulletin board operates V21 at 300 baud, 7 data bits, 1 start bit, 1 stop bit, and even parity. Transmission is in ASCII with the occasional refinement of Xon/Xoff protocol. Stick to this until you have the confidence to try anything fancier and faster. Be warned, though, that your phone bills will suffer.

We casually mentioned telecommunications programs. PIP is, of course, the CP/M program that serves for simple telecommunications. The authors have used PIP successfully to transfer programs (in '.HEX' form) from California to Suffolk but there are programs that are purpose-built for the telecommunications.

Telecommunications programs generally operate a 'protocol' to make sure that data is sent when it can be received, and can be repeated if an error occurs in transmission. A file is sent by splitting it up into small units called 'blocks' or 'packets', and sending each in turn, repeating if there was an error in receiving it. Naturally the programs at either end of the phone line need to be able to operate the same protocol. The state of the art in telecommunication programs is UKM7, a UK modification and correction of MODEM7 by David Back. This uses a sophisticated version of the XMODEM protocol invented by Ward Christiansen in the States. This is available free of charge to members of the CP/M User Group (UK), in source and ready installed for the Amstrad CPC 6128/PCW 8256, and most other British micros. This program allows error-free transmission of any type of CP/M file including '.COM' files. As it uses the ubiquitous XMODEM protocol, it can be used for downloading compiled software from bulletin-boards. BSTAM is another widely used program is a commercial program and is expensive to buy. It is very simple to use, possibly more so than UKM7, but the protocol is not quite so good as that of UKM7. It is not available installed from the Manufacturer, but a patch for the Amstrad machines is available from the CP/M User Group (UK). Ascom is another commercial program that is able to operate the XMODEM protocol but it does not have enough advantages over UKM7 to make it worth while. MOVE-IT is another program. It has the advantage of being able to 'take control' of the other micro so that one can specify what files one wants to receive from the remote computer. All the communication protocols and programs mentioned so far are for CP/M use, and it is not really possible to communicate with other type of computer, such as UNIX, or to a mainframe. KERMIT gets round these difficulties by a very elaborate protocol that 'negotiates' between the two micros the best way of communicating. This is because of the wide disparity in facilities available to a serial channel in different types of mainframe. KERMIT is in the public domain and, again, available from the user group. It will talk quite happily with another CP/M micro but the protocol is a bit like overkill for such a use.

It surprises me that, with so many CP/M users in the UK, and the presence of huge amounts of software with publicly owned copyright, that there is so little interest in telecommunications. I hope that a more encouraging attitude from Telecom, the arrival of the cheap modem, and the wider availability of free comms software will encourage the average CP/M user to dabble. It would save them an awful lot of problems.

## CHAPTER 6 – Writing CP/M software

CP/M software is easy to write. If this was not so, then CP/M would never have become so popular as it is. The operating system supplies all the fundamental facilities for a disc-based operating system and character-oriented i/o. Graphics and full-screen handling has to be done directly to the hardware or to GSX, the operating-system extension.

The BDOS is accessed through a vector at location 0005H in the first page of memory. For details of the way this is done, and the way that one interfaces to CP/M in assembler, see chapter 7. In this chapter, we will give hints about the more common requirement, using languages with CP/M and accessing the BDOS from a high-level language. We provide a resume of the following languages:

- 6.1.1     The C language
- 6.1.2     The BASIC language
- 6.1.3     The PASCAL language
- 6.1.4     The PLI language
- 6.1.5     The FORTH and STOIC languages
- 6.1.6     The LOGO language
- 6.1.7     The FORTRAN language

Firstly, choose your language with care. In the long run, it is always worth investing in the best, even though the cost of the software can approach the cost of the hardware. There are two types of language. There are the traditional languages such as Pascal or BASIC and there are the application languages such as DBASE II or 'Sensible Solution'. The application languages are generally more useful for developing applications for commerce. The use of the traditional languages gives you more versatility at the price of development time. As the application languages generally require no special interface into CP/M, we will not discuss them in this chapter.

The authors of this book program in a lot of languages, tending to choose the language to suit the task. As we are both professional systems programmers, we generally have to program directly in assembler language. RMAC and LINK are obligatory tools for the assembly-language programmer and they are both supplied with CP/M Plus. We actually prefer M80 from Microsoft, as it will assemble both Zilog and Intel Mnemonics. Once you have a good text editor that suits you, you are fully equipped.

There are an abundance of languages to choose from, and a number of implementations of each. They range in price from the expensive, to the free.

### 6.1.1 The C language

C is a much-vaunted language. We use it a great deal as it provides the means to write portable software. It is not as good as the popular press and certain computer scientists would have one believe. It is not nearly as efficient as assembler code, it tends to code up awkwardly on the Z80 chip, due to its reliance on passing parameters and automatic variables on the stack. It is easy to make a mistake, and it is difficult to trace the elusive bug. Unlike the strongly typed languages such as Pascal or PLI, the compiler lets through all sort of errors without complaint. Although it is nice to have the freedom of C, it can waste an awful lot of time. There are a great number of utilities and libraries in C to help the programmer, but C really comes into its own only on the 16-bit micros. It is not really the computer language for the beginner. The language has never had a rigorous definition and software that has been written for one particular compiler may run on another, but it may fail for subtle reasons. Some implementations are infuriatingly idiosyncratic, and often leave out such essential features as an ability to deal with real numbers.

### 6.1.2 The BASIC language

BASIC is the source of undeserved derision from microcomputer snobs. There is nothing wrong with BASIC, particularly as a beginners language, and one can do wonders with it. I know of commercial software houses that program in nothing else. The provision of a resident BASIC on the Amstrad machines is a useful feature, but the danger is that the user feels compelled to use it, because it is there. BASIC suits most people very well but it does not suit everyone. BASIC was designed as an interpreter that could be used to teach students. It was very simple and effective. Unfortunately, because it was so easy to write such an interpreter so easily in the restricted memory of the early micros (as little as 3K) it became THE language for micros. The extra features were grafted on with little real feel for the original language. Such things as file handling, memory and port access were grafted on in a unique manner for each implementation. The result was a mess, with major problems in making BASIC programs, written for one computer, work on another. Interpreted BASICs are slow, some very slow. Basic compilers produce quick code but they are expensive.

### 6.1.3 The PASCAL language

Pascal is an excellent language, with a number of good implementations about. If it is properly implemented, a Pascal Program is likely to work properly if it compiles. All those errors that are due to typing errors, or confusions in variables are sorted out by the rigorous checks made by the compiler. The Pascal language encourages structured programming. Like BASIC, however it is strictly a teaching language and is really unsuitable for serious use. Its file handling is primitive, it cannot manage dynamically sized objects such as strings, and it has only a dim idea of devices and I/O redirection. For certain uses it is supreme, and is a natural language for illustrating computer algorithms. The best implementation is probably ProPascal, but TurboPascal is cheaper. TurboPascal does not manage to produce '.REL' files so does not encourage real modular programming. Long programs need to be compiled in toto every time a program is changed. If a program is reasonably sized, then this is not a real problem. The beauty of TurboPascal is the built-in text editor that makes program development as easy as BASIC.

#### 6.1.4 The PLI language

PLI is a huge mainframe language that has been implemented very nicely under CP/M. PLI-80 was written by Gary Kildall, who wrote CP/M itself. There are versions that will run under MSDOS/PCDOS and Concurrent CP/M. These are nearly identical to the 8-bit versions, so there are no problems in software portability. PLI-80 is actually an implementation of a well-defined subset of PLI (subset G) and, in this relatively slimmed-down form, is an excellent language that combines the best qualities of Pascal, C, Fortran and Cobol. Although it is rather short on control structures, it has all the features that a programmer could desire, and is just as suitable for systems work as it is for commercial work. Code written for PLI can be just as compact as that written in C and certainly runs as fast. It is a well defined language that is strongly typed. It is not the easiest to learn but is a good all-rounder. Its rather high price deters any wide acceptance.

#### 6.1.5 The FORTH and STOIC languages

FORTH is an interesting language, as it shares none of the common features of the 'Algol family' of C, Pascal and PLI. It is probably the easiest language to move onto a new chip and generally appears first when a new chip is launched. It is a very simple and primitive language, to the point where it can be microcoded into the CPU itself. The Forth primitive 'words' become the real machine language. Forth is a language that addresses itself to a hypothetical stack-oriented machine. What this means is that the programmer has only to imagine that he is programming this chip, whatever the real chip happens to be. The task of the kernel of the Forth language is to make the real hardware act the same way as the hypothetical one. Once it has done so, the rest of the compiler/interpreter is easy, as is written in the kernel language itself. Forth suffers most from its lack of the in-built ability to handle real numbers. Its file handling is primitive and rather bizarre on first sight. It is, however, easy to develop difficult control software in Forth as each subroutine can be exhaustively tested. Forth has a devoted following. Its odd syntax and deviant ways should not put you off, it is a useful tool. There is a rather more rational dialect of FORTH called STOIC, which is worth trying.

#### 6.1.6 The LOGO language

LOGO is an alternative learning language to BASIC and PASCAL. It has some useful features, taken from its parent language, LISP. It is an interpretive language, and therefore is rather easier than Pascal. Its chief use is as an interactive learning language for children. Contrary to popular misconception, Logo is not the nearest computer language to natural language, and it is possible to write startlingly obscure code in Logo. It is not suitable for writing useful programs on micros due to its slowness in execution and its bulk. The most vaunted feature is the 'Turtle Graphics' which can produce delightful graphic pictures on screen. 'Turtle Graphics' however, are not unique to Lisp, and have been implemented in other languages. The important combination is 'Turtle Graphics' and interactive (or interpretive) language. Logo is excellent for controlling real turtles, which are floor or table-based wheeled robots with pens attached, and can provide almost the entire maths syllabus in a Primary class. Unfortunately, the Local Education Authorities are unmotivated to explore the Logo/floor-turtle combination and are unable to fund the hardware costs, but there is hope for the future here.

### 6.1.7 The FORTRAN language

FORTRAN is the granddaddy of computer languages. It is a compiler from which BASIC is descended. There exist a huge number of ForTran programs on mini- and mainframe- computers and these can be persuaded into running on a micro, particularly if a CP/M ForTran compiler is used in collaboration with an overlay linker. Most of the 8-bit Fortran compilers are rather old and have had little development work in recent years. There exists one excellent implementation, ProForTran, which is robust and efficient. ForTran is still highly regarded, but is superseded for most purposes by more modern languages. There exists a preprocessor, called RATFOR, which converts a language similar to C into ForTran.

### 6.2 Accessing operating system environment from a high-level program

When writing programs in a high-level language, CP/M is not much in evidence. A Pascal environment, for example, should be the same in UNIX, CP/M, MSDOS, CTOS or whatever. Once one has mastered the niceties of running the compiler and linker, it should not matter what operating system is running. Things are actually never that simple, and programs that were written for one version of a language under one operating system usually need a great deal of massaging to run on another. A large program will often need to address itself to the operating system directly, in order to ascertain certain facts about the hardware and filing system. How many files are on disc? how many drives are attached and which ones are they? What files are on disc? What devices are available? Has the user pressed a key? What is the time? What user area are we in? How much room is left on disc?

A program may need to chain to another or load overlays. At this point, the program will have to address itself directly to the BDOS, the kernel of CP/M.

Lets start with a simple example. We want to know whether a file exists. This is a simple routine. We assume a call 'bdos' that passes, as its parameters two variables, which will be put into the BC register pair and the DE register pair. We also need a routine to format a filename into an FCB, called 'fcbinit'. (writing such a routine is unnecessary as CP/M does it for you. We will see how this is done later).

```

/*-----*/
does_exist (filename) /*returns true if the file exists, otherwise returns
false */
char *filename;

{
char fcb_address[36];

fcbinit ( filename ,fcb_address); /* initialise our pattern */
bdos (26, BASE + 0x80); /* ensure default DMA after read/write */
return (bdos (SEARCH_FIRST, fcb_address)!= 0xFF);
}

```

Now this routine should cause no panic. As there may be subtle differences between C compilers, it is not worth displaying the construction of the BDOS function. It is best written in assembler and put into the library file if it is not there already.

Let us take a second example (in C). A program has been given an ambiguous filename as a parameter, as the user wants the program to process a number of files. We are faced with the daunting prospect of finding out what is on the disc and assembling a list of files conforming with that particular AFN (ambiguous filename) ie, unravel an AFN into a list of UFNs (unambiguous file names). No languages will do this for you.

There are two BDOS calls that are required, SEARCHFIRST and SEARCHNEXT. (see chapter 7). The SEARCHFIRST call presents you with the first occurrence of a UFN corresponding to an AFN. Then the subsequent ones must be sought with SEARCHNEXT. Essentially we need a routine that presents the AFN to the operating system once with SEARCHFIRST, and then loops round repeating the operation with SEARCHNEXT until the operating system flags the fact that no more occurrences exist. We need to squirrel away each file that the operating system finds, building up a list that the program can subsequently use. It is best to build up the list in one fell swoop, as one gets odd things happening if one makes other disc-based BDOS calls between SEARCH calls.

so we start with a vague notion of what is required:-

```
if (a_UFN_exists_for(AFN) Squirrel_away (the_first_file(AFN));
while (more_UFNs_exist_for(AFN)) squirrel_away(the_next_file(AFN));
```

Now, under this scheme, the procedures 'a\_UFN\_exists\_for' and 'more\_UFNs\_exist\_for' return booleans dependent on whether the search was successful, whereas the\_first\_file and the\_next\_file return the address of the file. squirrel\_away just adds the file to a list somewhere.

There are improvements to be made: If no UFNs exist on the disc, then we do not wish to execute the second line. Although the code above is clear, it is awkward to implement because the BDOS call returns both the information as to whether the file exists, as well as information about where the UFN is.

Let us assume that we have a library utility called 'bdos' that makes a BDOS call using the parameters:-

```
unsigned char function_number;
int bdos_parameter;
```

and returns the parameter that CP/M returns in the A register.

Our first problem in coding up the procedure is to convert the filename that is held as an ASCII string, into a File Control Block. CP/M, unlike MSDOS 2.x, cannot recognize the file held as an ASCII string. Often, it is not necessary, as the CP/M loader attempts to format the first two words in the parameter tail of a command line into FCBs, putting them at 005CH and 006CH in memory. In C, the 'argv, argc' mechanism reads the same uninterpreted command tail from 0080H, putting them in an array, word by word, so we would have to cheat. Fortunately, CP/M Plus has a BDOS call to do it for you. We will call the procedure using this call 'fcbinit' in this example.

```
fcbinit ( the_afn, fcb_address); /* initialise our pattern */
```

This would put the resulting FCB in the character array 'fcb\_address'

Now we make a call to set the DMA address to the correct default DMA position in the first page of memory. This is only necessary for early releases of CP/M.

```
bdos (SET_DMA, BASE+0x80);          /* ensure default DMA */
```

Now it is useful to remember which drive we are about to search. If the AFN actually specified the drive, The search will be on this drive, otherwise it will be on the default drive. We shall call this variable the 'unit'.

```
unit= (fcb_address[0] ? fcb_address[0]-1 : bdos (GET_DRIVE, 0) );
/* ascertain which drive we are doing the search on */
```

ie: if the first field of the FCB is non-zero, then it specified the unit (in the range 1..n), otherwise it is the default drive, which can be ascertained by means of a BDOS call 'GET\_DRIVE'. (in the range 0..n-1). whichever the unit was, we convert it to range from 0..n-1.

Having done this initial housekeeping, we leap into the main loop, using the loop initialisation feature of C to make the initial SEARCH\_FIRST.

```
for (dmapos= bdos (SEARCH_FIRST, fcb_address); dmapos<4 ;)
{
    squirrel_away(BASE + 0x80 + dmapos*32)); /*store the filename away*/
    dmapos= bdos (SEARCH_NEXT, fcb_address); /* and go for another */
}
```

If dmapos is returned OFFH, then it flags the end of the search, otherwise it records the relative field within the directory window at BASE + 0080H. We should, perhaps, now flesh out the squirreling away activity. We need to have a structure with forward links that allow sequential access from the start of the list:

```
struct next_filename      /* a linked list of filenames in ASCII format */
{
    char filename[15];     /* a filename */
    char *next_link;       /* pointer to next */
};
```

and we will need a starting point in the list and a pointer to it.

```
struct next_filename root, *fname;
```

we would access each name for processing through a procedure such as:-

```
char *the_next_file() /* returns a pointer to the next file and bumps the
                        file structure pointer */
{
    fname=fname->next_link; /* make fname point to the next link in the list */
    return(fname->filename); /* and return, pointing to the filename */
}
```

A problem we will need to confront is turning an FCB back into a filename. Our list of files will be required in ASCII string format (null terminated) as the C library functions require ASCII filenames. This will be done by a separate procedure which we will call 'hackname'. In our version, we access the global called 'unit' to get the drive, though this might be better passed on the stack along with the other parameters.



```

/*-----*/
hackname (destination, source) /* make a string representing a file name from a
                                directory FCB put it in 'destination' */
char *destination, *source;

{
    int i, j=2;
    /* insert the drive descriptor at the start, followed by a colon.*/
    destination[0]= unit + 'A'; destination[1]=':.';
    /* get the filename until it hits the trailing spaces or is eight chars long*/
    for (i = 1; ((i < 9) && (source[i] != ' ')); i++)
        destination[j++] = source[i];
    /* put in the '.' separator if a filetype was specified */
    if (source[9] != ' ') destination[j++] = '.';
    /* and now do the filetype */
    for (i = 9; ((i < 12) && (source[i] != ' ')); i++)
        destination[j++] = (source[i] & 0x7F);
    /* lastly, null-terminate the resulting string */
    destination[j] = '\0';
}

```

The actual squirreling is done by grabbing memory from the heap, as we do not know in advance how long the list will be. This is probably best done within the loop, as it is the only place within the program where we will require to add to the list. Let us see now how the module looks:

```

#define SEARCH_FIRST      17
#define SEARCH_NEXT      18
#define SET_DMA          26
#define GET_DRIVE        25

/*----- Structures -----*/
struct next_filename      /* a linked list of filenames in ASCII format */
{
    char filename[15];      /* a filename */
    char *next_link;        /* pointer to next */
} root, *frame;

/*----- Variables -----*/
int unit; /* the drive 0-n referenced by the ambiguous file (AFN) */
int file_count = 0; /* the number of files found so far */

/*-----*/
char *the_next_file() /* returns a pointer to the next file and bumps the
file structure pointer */

{
    frame=frame->next_link;
    return(frame->filename);
}

/*-----*/
unravel_afn(the_afn) /* unravel the ambiguous filename and put it in the
chain of filenames */
char *the_afn;

{
    char *malloc ();        /* it returns a pointer */

```

```

register unsigned char dmapos; /* the index into the directory window */
char fcb_address[35];        /* put the FCB here */

fname=&root;                  /* initialise the list to the root */
fcbinit ( the_afn, fcb_address); /* initialise our pattern */
bdos (SET_DMA, BASE+0x80);    /* ensure default DMA */
unit= (fcb_address[0] ? fcb_address[0]-1 : bdos (GET_DRIVE, 0) );
/* ascertain which drive we are doing the search on */

/* now read in all the files */
for (dmapos= bdos (SEARCH_FIRST, fcb_address); dmapos<4 ;)
{
    file_count++;             /* keep a tally of the number read in */
    fname=(fname->next_link=malloc(sizeof(root))); /* make the link */
    hackname (fname->filename, (BASE + 0x80 + dmapos*32));
    dmapos= bdos (SEARCH_NEXT, fcb_address);
}

/*-----*/
hackname (destination, source) /* make a string representing a file name from a
                                directory FCB put it in 'destination' */
char *destination, *source;

{
    int i, j=2;
    /* insert the drive descriptor */
    destination[0]= unit + 'A'; destination[1]=': ';
    /* get the filename */
    for (i = 1; ((i < 9) && (source[i] != ' ')); i++)
        destination[j++] = source[i];
    if (source[9] != ' ') destination[j++] = '.';
    /* and the filetype */
    for (i = 9; ((i < 12) && (source[i] != ' ')); i++)
        destination[j++] = ( source[i] & 0x0F );
    /* null-terminate it */
    destination[j] = '\0';
}

```

Of course, further improvements are required. The 'malloc' call is unprotected, in that, if no memory is available, and 'malloc' returns a zero, then the program would crash. The function 'the\_next\_file' might free the structure element it scraps. Once you have a module that works well it can be used and reused over, and over again.

Let us take another example. Suppose we want to know what drives are logged-in, so we can search on all for some file (such as our adventure data files.) We can use the CP/M call LOGIN\_VECTOR.

This call returns the bitmap of all the logged-in drives. Using our previous 'bdos' procedure, we can simply do a

```
login=bdos(LOGIN_VECTOR,0);
```

but the login bitmap is a rather obscure if you are unfamiliar with this type of bit-array. To make things easier, we design a function that returns the drive of the next logged-in drive by name (A..P). As the return code of the bdos call is actually a sixteen-bit bitmap in HL rather than A, in later

versions of CP/M, we need to use a modified version of the call, 'bdoshl' returning the value in HL after the bdos call.

```
#define FALSE 0
#define LOGIN_VECTOR 24
int login;
char drive_count;

get_login() /* gets the login bitmap and initialise the number of
rotation done on that bitmap to zero */
{
    login=bdos(LOGIN_VECTOR,0); /* make the bdos call */
    drive_count=0;
}

the_next_drive() /* returns a character representing the next logged-in
drive. login must have been set up by a bdoshl call,
and drive_count initialized. Returns FALSE if no more
logged-in drives, otherwise the character representing
the drive. */
{
    while (!(login & 0x01)) /* test bit 0 in the shifted bitmap */
    {
        if (drive_count++ > 15) return (FALSE); /* if all done */
        login = login >> 1; /* shift the bitmap */
    }
    login = login >> 1; /* shift ready for the next call */
    return (drive_count++ + 'A'); /* and return */
}
```

These two procedures are then used for ascertain which drives are logged-in and allows such procedures as supplying sorted directories including all drives.

For our next example, we will choose a function that require both BDOS calls and data conversion. Imagine we want a utility that returns the Time and Date in proper UK format. This is nothing to be ashamed about; the usual computer date format matches impressiveness with indecipherability. To get the time from the machine, we need to make a BDOS call. The call fills in a table within the program area. The format of the date is not human-orientated. we have an integer representing the date in days since the start of 1978. We have the hour in 24-hour form in two packed BCD digits, and we have the minutes in the same form. From this, we must calculate the date, taking leap years into account.

The CP/M time structure (called TOD) is something like this:

```
struct {
    int day;
    char hour;
    char min;
    char sec; /* it cannot manage this field */
} tod; /* Structure passed to Cpm to fill in */
```

note that the sec (seconds) field is included for MP/M II and Concurrent DOS compatibility, but it is not filled in by CP/M Plus (another call is required: see chapter 7).

The actual call is very simple.

```
bdos(T_GET,&tod);    /* what is the time, please */
```

but there are a number of beads of sweat to get the date converted into something sensible. Ironically, the clock chips have improved so much that it is far easier to read such a chip directly than to get the CP/M time and date. However machines such as the PCW 8256 have no clock chip, and we want a portable CP/M program.

We have given all the code in the form of a program, though it is easily installed as a routine passing back a pointer to a string containing the nicely formatted date. It would be very smart for invoices or printouts.

```
/*----- Utility to give the time and day -----*/
*/

#define T_GET 105
#define CPM_START 1978 /* CP/M gives days as integers beginning jan. 1978 */
#define S_BDOSVER 12    /* find the bdos version */

/*--- Statics ---*/

int day_table[2][13] = {
    {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31},
    {0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}
}; /* No. of days in a month table, for leap and non-leap years */

char *dayname[] = /* the names of the days */
    {"Saturday", "Sunday", "Monday", "Tuesday",
     "Wednesday", "Thursday", "Thursday", "Friday"};

char *monthname[] = /* the names of the months */
    {"January", "february", "march", "april", "may", "june",
     "july", "august", "september", "october", "november", "december"};

/*-----*/
main() /* Make Cpm system call to get the system time/date structure */
{

    int hour, day, month, year; /* the integer values of the date components */
    int days_in_previous_years=0;
    int leap; /* is it a leap year? 0 or 1 */
    char *noon; /* points either to "am" or "pm" */
    char *ordinal_name(); /* this procedure returns "st", "nd", "rd" or "th" */

    struct {
        int day;
        char hour;
        char min;
        char sec; /* it cannot manage this field */
    } tod; /* Structure passed to CP/M to fill in */

    if (bdos(S_BDOSVER, 0) < 0x30)
    {
        printf("sorry, wrong version of CP/M");
    }
}
```

```

    exit(1);
}
bdos(T_GET,&tod); /* what is the time, please */

/* First find what the next year will be */
for (year = CPM_START; days_in_previous_years < tod.day; year++)
    days_in_previous_years += its_a_leap (year) ? 366:365;

/* now back off one */
days_in_previous_years -= its_a_leap (--year) ? 366:365;

/* Now find the current month and day in the month */
day = tod.day - days_in_previous_years; /*day in current year */
/* and whether it is a leap year */
leap = its_a_leap (year) ? 1 : 0;
/* repeatedly add in each month */
for (month = 1; day > day_table[leap][month]; month++)
    day -= day_table[leap][month];
/* get the integer version of the hour */
hour=converted(tod.hour);
/* and set noon to point to "am" or "pm", abandon 24 hour clock nonsense */
if (hour > 12) (hour -=12; noon="pm"); else noon="am";
/* now print it out to taste */
printf ("the time is:- %d:%d %s, %s %d%ks %s, %d (%d/%d/%d)",
    hour,          /* the hour (1..12) */
    converted(tod.min), /* the minutes */
    noon,          /* either "am" or "pm" */
    dayname[tod.day % 7], /* the string for the day name */
    day,           /* the day of the month */
    ordinal_name(day), /* "st","nd","rd","th" */
    monthname[month-1], /* name of the month */
    year,          /* full year */
    day,           /* day of month */
    month,         /* month number */
    year-1900 );   /* truncated year */
}

/*-----*/
char *ordinal_name(number) /* returns the ordinal postfix for a number
    eg 1st 2nd 3rd 4th. We like user-friendliness dont we? */
int number;

{
    int units;
    static char *ord_string[] = {"th","st","nd","rd"};

    if ((units = number % 10) > 4) return("th");
    if ((number > 20) || (number < 10)) return (ord_string[units]);
    return ("th");
}

/*-----*/
converted(the_char) /* turn a BCD pair into an integer, returning the
    integer */
char the_char;
{
    return( (the_char & 0x0f) + ((the_char >> 4) * 10) );
}

```

```

}

/*-----*/
its_a_leap(our_year) /* returns true if the year passed to it was a leap year
                      otherwise returns false */
int our_year;

{
return ( !(our_year%4) && (our_year % 100) || !(our_year % 400));
}

```

In our next example in C, we will attempt to ascertain from what drive the program was loaded. If, for example, we were operating a program, such as an adventure program or one using overlays, we would need to know which drive to get the ancilliary files. One fault with current programs using overlays, such as Wordstar, is that it is not possible to run it of another drive other than the currently logged one (other than A). Wordstar locks up with an error.

In CP/M Plus, there is a byte held in location 0050H that is copied from the FCB of the program that was loaded. Like all FCBs, byte 0 has a value 1..16 corresponding to drive A..P. a value of 0 represents the currently logged drive.

The actual routine will be familiar from our first example:-

```

/*-----*/
char *boot_drive() /* returns a string representing the drive specification
                   of the drive from which the program was loaded. The program
                   should do a version check before doing this call */
{
static char drivespec[3]="A:";
drivespec[0] = 'A' + (*0x0050 ? *0x0050-1 : bdos(GET_DRIVE, 0));
return(drivespec);
}

```

Our next example provides a rather useless program, but it shows how to get hold of, and alter parameters that are inaccessible from elsewhere in the system. I used to hear the frequent plea from programmers that they wished to switch or read the print toggle (^P at the CCP) in order to send console output to the printer. The following example shows the way. Also accessible are such things as the setting for the temporary drive, the drive search chain, the current DMA address, the console dimensions (used by DIR), the redirection bitmaps, the date, and other facilities too many to enumerate here. (see the chapter on the BDOS on the SCB). There is also a small area of reserved memory that can be used for communication between programs, passwords, and so on. All these items of information are in the SCB that can be accessed and modified from a program. The SCB can be defined as structure in C as follows:-

```

struct scb_stuff
{
unsigned char first_reserved[5];
unsigned char version;          /* version of the BDOS */
unsigned char user_flags[4]; /* define these as you wish */
unsigned char second_reserved[6];
unsigned char error_code[2]; /* used to signal errors */
unsigned char third_reserved[8];

/* console characteristics */
char width_of_console;

```

```

char xpos;           /* the position of cursor */
char length_of_console; /* number of lines */

unsigned char fourth_reserved[5];
/* our device redirection flags for each of the five logical devices */
unsigned int conin_redirection;
unsigned int conout_redirection;
unsigned int auxin_redirection;
unsigned int auxout_redirection;
unsigned int listout_redirection;
char page_mode; /* do we scroll continuously or stop at each page */
unsigned char fifth_reserved;
unsigned char echo_backspace; /* do we echo destruc backsp.*/
unsigned char echo_delete; /* or echo delete */
unsigned char sixth_reserved[3];
unsigned int console_mode; /* see programmers guide */
unsigned int seventh_reserved;
unsigned char delimiter; /* string delimiter (def. $)*/
unsigned char echo_to_printer; /* ^P flag */
unsigned char reserved;
char *scb_pointer; /* where are we? */
char *dma; /* DMA address */
unsigned char current_disc;
unsigned char eighth_reserved[5];
unsigned char user_number;
unsigned char ninth_reserved[5];
unsigned char multisector_count; /* current multisector count */
unsigned char error_mode;
unsigned char search_chain[4]; /* as set by SETDEF */
unsigned char temp_drive; /* as set by SETDEF */
unsigned char error_drive; /* last drive of error */
unsigned char tenth_reserved[5];
unsigned char bdos_flags; /* see programmers guide */
/* the date and time */
unsigned int year;
unsigned char hour;
unsigned char minutes;
unsigned char seconds;
/* where the base of common memory is */
unsigned char *base_common;
unsigned char eleventh_reserved;

) *scb;

```

Now, the information in this CP/M structure is very valuable, and enables the programmer to manage interesting features. Obviously, the programmer must look at appendix A in the Programmers Guide for a complete description of each variable.

We need to initialise the SCB pointer to the right place. To do this, we must read an undocumented variable in the SCB using the BDOS call 'GET/SET System Control Block'. This undocumented variable actually points to the base of the SCB. To do this we have to define a second structure:-

```

/* the structure of the parameter block expected by BDOS call 49 */
struct scbpb
{
    char offset; /* which byte to set? */

```

```

char set;          /* do we want to set or read it */
int value;         /* what value to set it to */
) our_scbpb;

```

We can now get the pointer quite simply:-

```

/* let us find out the address of the SCB */
our_scbpb.set = 0x00;
our_scbpb.offset = 0x3A; /* pointer to base of the SCB */
scb= bdoshl(49,&our_scbpb);

```

As an example of using this information, here is a routine to print out some of the variables:-

```

printf(" CP/M Plus version %d",scb->version>>4,scb-> version & 0x0f);
printf(" the echo delete flag=%dH, console mode=%dH.",scb->echo_delete, scb->console_mode);
printf("\nString delimiter =%c and the echo-to-printer flag=%dH.",scb->delimiter,scb-> echo_to_printer);
printf(" the SCB is at 0dH and the DMA address is 0dH.", scb->scb_pointer, scb->dma);
printf("\nThe current disc is %c:, the user number is %d.",scb->current_disc+'A',scb->user_number);
printf("\nThe multisector count is %d, the error mode is %dH",scb-> multisector_count,scb->error_mode);
printf("\nThe drive search chain is %c:, %c:, %c:, %c:. (@ means default drive)",
scb->search_chain[0]+'@',
    scb->search_chain[1]+'@',
    scb->search_chain[2]+'@',
    scb->search_chain[3]+'@');
printf("\nThe temporary drive is %c: (@ means default drive).",scb->temp_drive+'@');
printf("\nThe last error occurred on drive %c: (@ means no error).",scb->error_drive+'@');
printf("\n console width is %d, cursor column is %d, console length =%d.",
    scb->width_of_console,
    scb->xpos,
    scb->length_of_console);
printf("\npage mode is %dH",scb->page_mode);
printf("\nRedirection flags are as follows:\n");
printf("conin=%dH, conout=%dH, auxin=%dH, auxout=%dH, listout=%dH.",
    scb->conin_redirection,
    scb->conout_redirection,
    scb->auxin_redirection,
    scb->auxout_redirection,
    scb->listout_redirection);
printf("\nBdos flags are %dH",scb->bdos_flags);
printf("\nYear is %d (days since Jan 78), hour is %dH, minutes =%dH, seconds =%dH.",
    scb->year,
    scb->hour>>4,
    scb->hour&0x0F,
    scb->minutes>>4,
    scb->minutes&0x0F,
    scb->seconds>>4,
    scb->seconds&0x0F);
printf("\nThe base of the common block of memory is 0dH",scb->base_common);

```

Interesting, but not of immediate use. Why not use the drive search path to scan drives for data files? Shouldn't you use the temporary drive for your temporary files? Once you have the time, it would allow easier program access to the time.

Some of these variables can be altered:-

```

user_flags[]           These can be altered freely for inter-program

```



	communication.
error_code[]	These transmit an error code to a chained program.
width_of_console	Can be altered to affect DIR displays.
length_of_console	Can be altered to affect paging.
conin_redirection	Can be altered to change redirection.
conout_redirection	Can be altered to change redirection.
auxin_redirection	Can be altered to change redirection.
auxout_redirection	Can be altered to change redirection.
listout_redirection	Can be altered to change redirection.
page_mode	Can be altered to page or not
echo_backspace	Can be altered to echo instead of destructive bspace
echo_delete	Can be changed to echo instead of delete
console_mode	Can be used to change the console mode
delimiter	Can alter the terminator for string output
echo_to_printer	1 to send to list device as well as console, else 0
multisector_count	Current multisector count
error_mode	The way that the BDOS handles errors
search_chain[]	Can be set as by SETDEF
temp_drive	Can be set as by SETDEF
year	May or may not set the clock!
hour	
minutes	
seconds	

The SCB provides facilities and information that cannot be got any other way. We will not spell them all out, as one must leave something up to your imagination. (The SCB is described in much more detail in the next chapter)

One question we are often asked is how one obtains the size of a file. This is easily enough done. We will take the opportunity of showing how to format a filename into an FCB ('fcbinit'). We need a simple structure that we can declare as:-

```
char *pfc[2];
```

and another to describe a file control block in its simplest terms (just for this routine!)

```
struct {
    char qmstuff[33];
    long int sectors;
}; our_fcb;
```

Now, armed with our two structures we can perform the operation:-

```
/*-----*/
get_size(filename)      /* returns the size of the file in sectors */
char *filename;
{
    our_fcb.sectors = 0;      /* not really necessary if file exists */
    pfc[0] = filename;      /* initialise things to format the FCB */
    pfc[1] = &our_fcb;      /* tell the BDOS the address of the FCB */
    bdos(152, &pfc);         /* format the FCB */
    bdos(35, &our_fcb);      /* and find out the size */
    return(our_fcb.sectors);
}
```

In practice, we would make the structures local to the function. Here is a similar routine to return the size in 'K'. We will use our routine 'fcbinit' instead.

```

/*-----*/
size_of_file(filename) /* returns the size of the file in K */
char *filename;
{
    struct {
        char cpmstuff[33];
        long int sectors;
    }; our_fcb;
    our_fcb.sectors = 0;          /* not really necessary if file exists */
    fcbinit( filename, &our_fcb);
    bdos(35, &our_fcb);          /* and find out the size */
    return((our_fcb.sectors >> 3) + (our_fcb.sectors & 0x07 ? 1:0));
    /* return K rounded up */
}

```

### 6.3 Programmers nostrums

When writing programs under CP/M, there are certain bits of advice that we have learned from experience. We offer them in no particular order of importance. One should remember that there is no 'royal road' to writing programs. If rules were never broken, the art of programming would never progress. Programming is a hybrid between art and engineering. The intuitive should go hand in hand with the scientific.

#### Programmers Nostrums

- 1/ If a program expects parameters, and the user supplies none, incorrect ones, or the wrong number, then provide an on-screen explanation of what the program does, and what parameters are required.
- 2/ Provide sensible defaults. If, for example, you write a program to convert a '.COM' file into a '.HEX' file, then you can assume that, unless the user specifies otherwise in the command line, he will want the '.HEX' file named to the same filename as the '.COM' file but with a '.HEX' filetype.
- 3/ Always document/comment source files properly. Even if no one else sees your work, it saves valuable time later on when you cannot remember why you wrote a particular bit of code, and what it does. If there is more code than comments, a program is underdocumented.
- 4/ Write the program documentation before you write the program.
- 5/ Use long, descriptive, symbols for such things as functions, procedures or variables. Long labels may bulk up the source, but do not affect the program size.
- 6/ Try to use existing conventions in programs. For example, if you are using 'conditional switches' to allow the user to define options in the way a program works, then try to use the normal conventions. (in CP/M Plus, the list of switches are usually enclosed them in square brackets. Where values or symbols are assigned in the parameter line, then it is by an normal assignment

statement. In C, they are preceded by '-'.) One of the problems of CP/M, is the inconsistency in command line conventions. ASM.COM, for example, uses the filetype field for assigning the location of associated files, whereas MAC uses the 'S' sign.

7/ Do not get carried away by the idioms of a particular language to write what looks like compact code. Only in an interpreted language is there much gain, and often, the length of the compiled code is quite unaffected by such feats of programming gymnastics. The result is too often dense impenetrable, and unmaintainable code. The C Language provides the greatest opportunities for excesses.

8/ Before starting to write a program, check to see if someone has already done so, and if it is in the CP/M User Group UK. library. If it is, then obtain it, improve it, and resubmit the results.

9/ Never write the same bit of code twice. Keep a library of the source code you write and reuse it. If your code is modular and disciplined, and you have a decent text editor, this will be easy.

10/ If you are writing in Assembler, then decide on a convention of passing parameters between subroutines and stick to it. Generally, it may be worth using the same conventions as your favourite compiler so that you can link them in together. There are three main methods, putting parameters on the stack, stuffing them into as many of the CPU registers as you can manage, or passing a pointer to a data area or table. One can, of course, place parameters in global variables and thereby avoid passing parameters as such, but the resulting code will be neither recursive nor re-entrant.

11/ Keep the user interface simple and clear. Do not design the user interface for programming convenience, but for the users convenience. All those slick pop-up or pull-down menus will not impress the end-user if, at another point in the program, he is required to hold down 'Control', 'Shift', and A at the same time.

12/ Make your error messages as long and explanatory as possible. Remember that you have all the TPA to use. Messages such as 'Fatal Error' merely cause alarm and despondency amongst the unwary and do not impress other programmers at all.

13/ Use BCD for real numbers in preference to floating-point numbers, especially for commercial work. BCD has more accuracy.

14/ When you have, or are operating on, data files, then keep the data in ASCII format if at all possible. It makes the inspection, maintenance, transmission or restoration of the data far easier. It also helps when debugging the program.

15/ When the program is up and running, then get someone else to test it. Programmers cannot test their own code. They are too gentle with it and unconsciously avoid the 'dodgy' parts of a program. Testing a program to destruction conflicts with the creative instinct. The best program testers get a malicious satisfaction from doing so. As children, they broke other children's Lego models, and used their train sets for causing train crashes. Such people are invaluable.

16/ There is no such thing as a bug-free working program. Programs are like tramps beds. There are always a few bugs lurking.

- 17/ When you alter a program, make sure you record the fact and the date in a comment within the code. Document your work in the header to the program.
- 18/ Develop your programs in a modular style. It is important for morale to get something running relatively quickly. Build on the skeletal structure, adding flesh to the bones. Do not attempt to build a program like Frankenstein built his monster; ie. trying to build the whole thing in one go, and then bringing a large uncontrollable, and vengeful, monster into life.
- 19/ If your program fails to work, resist the temptation for blaming the problem on the operating system or a bug in the compiler. It is far more likely to be a bug in your program, incredible as it might seem to you.
- 20/ The person who has to use a program is likely to be a creature of great wisdom. If he or she criticizes your program, it is very likely to be just and valuable criticism. No programmer can stand back from his program and view it objectively.
- 21/ It is easier to rewrite a program properly, than to attempt to modify or improve one that is poorly constructed or documented.
- 22/ Always use a dedicated application language in preference to a general-purpose language, where the former is suitable. DBASE II is likely to be a better and more profitable vehicle for an application than Pascal or C.
- 23/ Where the operating system provides a facility, use it. CP/M Plus provides line entry with editing and the provision of default strings. It is likely to be better than your version, so use it. Random access files, multi-sector i/o, RSXs, string output and the like, are all there to be used.
- 24/ The best way of improving the speed of a program, or making it more compact, is by improving the algorithm, (the way that the program, or its procedures, work) rather than rewriting an inefficient algorithm in assembler, or attempting trick code.
- 25/ The use of self-modifying code in assembler programs can often provide compact or efficient code. It can also produce code that defies debugging or maintenance. It is also 'unROMable' and is unlikely to work in a real-time environment.
- 26/ If you are writing code, and find yourself writing code that doesn't seem to 'jell' or is awkward and unstructured, then you should take a deep breath and start again. You have gone wrong somewhere fairly fundamental. You not get out of such a problem by putting your head down and attempting to 'code your way out' of the problem. Best start again.
- 27/ If you are writing code where speed is critical, design the structure of the code in a slow interpretive language such as BASIC. This will highlight the problems and make the slow bits more evident. When it is working well, as quickly as you can make it, then rewrite the code in assembler.
- 28/ Always do a version check on the operating system at the start of a program if you are using features of the operating system that were not in the first release. Abort with a full error message if the wrong version is being used. (eg Random Access -not in CP/M 1.4, Multisector I/O, RSXs, overlay loading, chaining, time and date, etc.)

- 29/ Never use any part of the first page of memory for variables.
- 30/ Never introduce hardware dependencies if it can be avoided. For example: if accessing the serial port, use the AUX: device rather than accessing the port directly.
- 31/ Do not plagiarize other ideas. There is a saying in the computer industry, "the pioneers are the ones who get the arrows in the back." This may be partially true but not for Cecil Rhodes, Henry Ford, Edison, Bell, etc. Be creative, but be familiar with other peoples ideas. Another saying is "The way to see further is to stand on other peoples shoulders."
- 32/ A relocating assembler is worth far more than an absolute assembler. Macros are nice, but a minor convenience. With a relocating assembler, programs can be built out of modules, and need not be completely reassembled after every change. Modules are easier to edit, and can be collected into libraries.
- 33/ Avoid self-modifying code. There is nothing clever in self-modifying code.
- 34/ Always check the CP/M version number if using BDOS calls that are incompatible with CP/M 1.4.
- 35/ Check the top of memory against the requirements of the program.
- 36/ Close files that you have opened, even if you are only reading from the file. Never access a file after you have closed it.
- 37/ Do not access the allocation vector table directly, unless you have checked that you are using CP/M 1.4 or 2.2.
- 38/ Do not mix direct character output with any other BDOS I/O function to the same logical device.
- 39/ Avoid direct BIOS calls.

## CHAPTER 7 – BDOS & BIOS

The first part of this book has explored CP/M at a leisurly pace, but now prepare yourself for the helter-skelter. If your stomach is not up to it, now is the time to turn out the light and go to sleep.

This chapter details the Basic Disc Operating System or BDOS and the Basic Input and Output System or BIOS. The information is essential for successful assembly-language programming under CP/M, but it is also intended for the high-level language programmer who wishes to access the BDOS and BIOS, and so it details each BDOS Function. Those of you with no desire to grasp the finer details of a BDOS function might however gain from an insight into the structure and methodology of the CP/M environment. The chapter starts by looking at the computer as it would appear to a transient program loaded into the TPA. Later sections explain the operation of the BDOS.

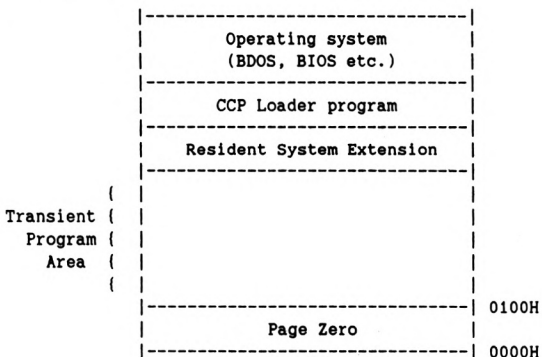
- 7.1 What is the TPA
- 7.2 Initialised data after loading
- 7.3 Essential Requirements of a TPA program
- 7.4 Terminating TPA program and return to Operating System
- 7.5 Operating system interface BDOS, BIOS and HARDWARE
  
- 7.6 BDOS calling conventions
- 7.7 BDOS character functions
- 7.8 BDOS disk functions
- 7.9 BDOS clock functions
- 7.10 BDOS System Control Block
- 7.11 BDOS system and miscellaneous functions
- 7.12 CP/M 2.2 BDOS compatibility
  
- 7.13 BIOS calling conventions
- 7.14 BIOS character functions
- 7.15 BIOS disk functions
- 7.16 BIOS system functions
- 7.17 Example of direct BIOS interface

The BDOS functions are grouped in tables at the end of each section to provide an ease of reference.

## 7.1. What is the TPA

TPA stands for 'Transient Program Area'. This is the area in the computer memory that is reserved for the execution of programs. The actual size of this memory area depends on what is left after the operating system and any resident programs are loaded.

The TPA always starts at location 0100h and its contents are uninitialized. The TPA is the memory space between Page Zero and the lowest location of the operating system, or any resident programs loaded below the operating system as shown in the following diagram:



The other areas in the CP/M memory map are:-

**Page Zero.**

This is the region of memory from the base at address 0000H to the base of the TPA at address 0100H. Page zero is 256 bytes, or 0100H, or 1 page long. Thus Page Zero is so named as it is exactly one page in length, it is the first page or, as computer numbering usually starts at 0, it is designated the Zero'th page. (Under MP/M, page zero is called BASE PAGE because the memory segment may not start at page 0)

**Resident System Extensions.**

These are a special type of program which can be added temporarily or permanently to the operating system to extend the operating system functions.

**The TPA Loader.**

This is used to load the transient program from its source (which is normally the disk) into the TPA memory area. The TPA loader only remains in memory whenever one or more Resident system program (RSX) are active.

The top of the TPA extends up to the address defined in the word value at bytes 6 & 7 in page zero. Usually, the TPA extends up to the lowest part of the operating system which is the BDOS.

The TPA is thus a free space of memory in the computer starting at address 0100H extending to the address in bytes 6 & 7. The size of the TPA will vary with the number and size of the Resident System Extensions (RSXs) that are active at the time.

Apart from RSX programs, all programs with very few exceptions reside in this TPA area, whether it is SHOW, WORDSTAR, BASIC, DBASEII, or any other program. The exceptions are programs such as the debuggers SID, DDT, ZSID which behave in a similar manner to RSX's. These utilities relocate to the top of the TPA and then reduce the size of the TPA area. Examples of RSX utilities are GET, PUT and SUBMIT. An RSX program file is identified by the loader as such because it has a 0C9H (RET) instruction as the first byte of the file, a highly unlikely occurrence in an ordinary '.COM' file!

When each RSX or debugger is loaded, the address in bytes 6 & 7 is altered to reflect the reduction in the TPA space, and to pass control to the RSX and/or debugger on each operating system call.

Each RSX will contain a link to the next RSX so that any operating system request will eventually reach the operating system BDOS. The following table illustrates the path together with the base address of a typical operating system and RSX configuration.

Base Address of system with RSX's and Debugger loaded

-----		
	Operating system (BDOS, BIOS etc.)	
+--->	CCP loader	F106H
+-----+	[FD0A]	
+--->	Multiple command line	
+-----+	[EC0A] RSX	EC06H
+--->	'PUT' RSX	
+-----+	[E70A]	E706H
+--->	'ZSID' Debug Utility	C500H
+-----+		
	Transient Program Area	
	Page Zero	
+-----+	[0006]	0000H
-----		



## 7.2. Initial data after loading

A program is loaded into the TPA area as a contiguous stream starting at location 0100H. The loading is performed by a LOADER module which is part of the CCP operating system module, but resides as an RSX (Resident System eXtension).

The CCP is the Console Command Processor and is best known for producing the A> prompt on the console screen. Typically the CCP first copies the LOADER module to the memory location just below the operating system BDOS, then the CCP requests a command from the console to be entered after the CCP prompt ( A> for example). If the command specifies a transient program (such as BASIC or WORDSTAR), then the LOADER module reads the disk file and copies the contents into the TPA. After loading the program, the loader passes control to the TPA by setting the program counter to 0100H.

Before the loader passed control to the TPA, the Page Zero is initialized, the System Control Block is initialized, and stack is provided with a return address. The processor registers are not initialized.

## Page Zero Initialization

0000H-0002H	WARM BOOT	Contains a jump instruction to the BIOS warm start entry point.
0003H	IOBYTE	not supported by CP/M Plus
0004H	DSK/USR	default drive/user when TPA loaded.
0005H-0005H	BDOS	Contains a jump instruction to the operating system, usually to BDOS.
0008H-000AH	RST1	Reserved for restart 1 instruction
0010H-0012H	RST2	Reserved for restart 2 instruction
0018H-001AH	RST3	Reserved for restart 3 instruction
0020H-0022H	RST4	Reserved for restart 4 instruction
0028H-002AH	RST5	Reserved for restart 5 instruction
0030H-0032H	RST6	Reserved for restart 6 instruction
0038H-003AH	RST7	Reserved for restart 7 instruction and for standard debug programs like SID.
003BH-004FH	scratch	Reserved area for customized BIOS (not normally used)
0050H	TPA:DSK	Disk from which program loaded (or zero for the default drive)
0051H-0052H	PSW1:PTR	Address of first password in default DMA buffer.
0053H	PSW1:LEN	Length of first password field
0054H-0055H	PSW2:PTR	Address of second password in default DMA buffer.
0056H	PSW2:LEN	Length of second password field
0057H-005BH	reserved	
005CH-007BH	FCB	Default file control block
006CH-007BH	FCB2	Second file name overlayed in the FCB for the first filename.
007CH	CR	Current position of default FCB area
007DH-007FH	RR	Optional random record number of default FCB area
0080H-00FFH	DMA	Default 128 byte disk buffer for transfers to and from the disk system.

WARM BOOT 0000H	The address at 0001H can be used to make direct BIOS calls for character functions
IOBYTE 0003H	The IOBYTE character-redirection feature of CP/M version 2.2 is not supported under CP/M 3.
DSK/USR 0004H	The default drive/user is only supplied to provide compatibility with programs written for CP/M 2. It is not maintained. New programs should use the BDOS calls.
BDOS 0005H	This is the interface into the CP/M Plus operating system. The address at location 0006H minus one defines the topmost address of the transient program area.
RST	The restart software instructions 1 to 7 make a call to the locations 0008H, 0010H, 0018H, 0020H, 0028H, 0030H, 0038H respectively. The restart location 0038H is also reserved as the default debug address for SID, DDT, and ZSID. (see note on hardware conflicts at the end of this section)
scratch 003BH	The area 003FH-004FH was used by the customized BIOS of early versions of CP/M. It is unlikely that any customized BIOS for CP/M 3 will use this area.

The memory locations from 0050H to 007FH are initialized by the CCP as follows:

TPA:DSK 0050H	A value of 1 to 16 identifies the drive from which the TPA program was loaded. A value of 0 identifies the default drive.
PSW1:PTR 0051H	If the first operand of the CCP command line tail contains a password, this address points to the start of the password field (see DMA). Otherwise both the address and the length are set to zero.
PSW2:PTR 0054H	If the second operand of the CCP command line tail contains a password, this address points to the start of the password field (see DMA). Otherwise both the address and the length are set to zero.
FCB 005CH	Default file control block initialized by the CCP from the first operand. If no operand exists, the FCB is initialized to drive zero and a blank name
FCB2 006CH	Default file name initialized by the CCP from the second operand. If no operand exists, the file name is initialized to blanks. This area overlays the default file control block. To use the file name in this area it must first be copied to another area before using the default file control block
CR 007CH	Before using the default FCB, a TPA program will usually set this location to zero.

RR           Before using the default FCB, a TPA program will  
007DH       usually set this location to zero.

DMA           The CCP copies the complete command tail into this  
0080H       128 byte area. The first byte contains the length of  
             the command tail, followed by the command tail  
             (if any). Because this memory area is the default  
             area for transfers of 128 bytes from/to the disk, the  
             CCP command tail should be copied to another area  
             before any disk activity is performed. Until the  
             BDOS is instructed to use another memory location for  
             DMA transfers, the BDOS moves 128 byte records to  
             this area during a read operation and BDOS copies 128  
             byte records from this area during a write operation.

#### Hardware conflicts with ZERO PAGE

With most computers there is no conflict between the reserved area in Page Zero and the hardware requirements. But on a few Z80 based computers the hardware can require a modified page zero.

The conflict arises because the Z80 processor can be programed to use the following locations for hardware interrupts:

0038H - Mode 1 maskable interrupt  
0066H - Non maskable interrupt

The Mode 1 interrupt is not compatible with the Intel 8080 interrupt, but was used to reduce the interrupt processing duration over the standard Intel 8080 designs. Fortunately most Z80 based computers use the much more powerful Mode 2 interrupts. Those computers that use Mode 1 interrupts CANNOT use the standard released CP/M because the restart address used by the Mode 1 is the same address as that used by the debug utilities DDT, SID and ZSID. Consequently these utilities and any other software which use restart 7 must be modified so that they use a different restart address before they can be used with any such computer. The Amstrad computers use Mode 1 interrupts, so the utility SID distributed with the computer has been modified.

The Non Maskable Interrupt uses location 0066h, which falls within the space reserved for the standard FCB. The standard FCB is used extensively by a wide range of CP/M utilities and programs, consequently it is not viable to modify the CP/M software to accommodate the Non Maskable Interrupt. CP/M cannot be used on any Z80 based computer which uses the Non Maskable Interrupt feature.

### 7.3. Essential requirements of a TPA program

A valid program must be able to execute its instructions without corrupting the memory area outside the TPA, and return control to the operating system when it terminates.

When control is passed to a TPA program, the only registers initialized are:

SP (stack) initialized to a stack area in the CCP loader  
IP (program) initialized to location 0100H

#### CCP LOADER stack area

Generally speaking, A CP/M Plus '.COM' file does not need its own stack as the one provided by the loader is adequate. This stack area is located at the top of the CCP loader. If RSXs are present, this stack is within the CCP loader and care should be taken to ensure that the limited stack space is not exhausted. Generally, if the loaded program contains all the data area required, that is it does not require any TPA memory space outside the space occupied by the loaded program, then the program can safely use the stack space provided by the CCP loader. If the program requires additional data space from the TPA, then the program might need to reserve its own stack area. Prior to using any such data and/or stack area, the program should check that the additional TPA memory space required is below the address contained in Page Zero at bytes 0006H-0007H. As SID and DDT set the program stack to 0100H, an area of memory that is highly likely to be corrupted by the first transfer at the default DMA position at location 0080H, it is important to set a program stack before a debugging session.

If the program is intended for use with other versions of the CP/M operating system such as CP/M 2.2, MP/M II or CP/NET, then the program should always allocate its own stack area and check the size of the TPA area.

According to the Digital Research Programmer's Guide, the stack area provided by the CCP loader is only 32 bytes. An examination of the CCP in the Appendix E shows a more complex stack area. The loader sets the stack to the base of the page containing the start of the BDOS. The loader stack extends up to offset 0BEH in the page below BDOS and this provides 66 bytes of space for the TPA stack. As the loader initializes the first entry in this stack with 0000H, these leaves 64 bytes or 32 LEVELS of stack (not 32 bytes as specified by Digital Research).

However part of this area is used by the CCP to pass the FCB of the opened file for the loader to load. If any TPA program also uses the loader to LOAD OVERLAY (BDOS function 59), then it should also use this 64 byte area for the opened FCB and stack to ensure that these important data areas are not overwritten by the loader when it loads a program.

The initialization of the TPA stack in the CCP loader with the first location set to 0000H enables any program to terminate with a RET instruction which will lead to a jump to 0000H and a warm boot. (Under CP/M 2.2, a RET would lead to an immediate re-entry into the CCP without a warm boot.)

The CCP loader stack thus contains 32 levels for the program use, of which one is required for any BDOS function, and from one to no more than 7 levels for any hardware interrupt.

## RESTRICTIONS ON Z80 INSTRUCTIONS AND REGISTERS

The TPA program should avoid any instruction which may conflict with the operating system. The following instructions and registers may cause problems:

REGISTERS IY, IX, I, R and the alternate register set are not available on all CP/M systems and should be avoided if the programs are to be portable across CP/M computers.

REGISTERS I, R should not be used.

ALTERNATE REGISTERS are provided for fast interrupt response, they should only be used when interrupts are disabled.

INDEX REGISTERS IX and IY are used on some CP/M systems by the BIOS, and to maintain their integrity, they should be saved (on the stack) prior to function calls to either the BDOS or BIOS.

INSTRUCTION HALT should never be used in a TPA program.

INSTRUCTIONS EX AF,AF' and EXX should only be used when interrupts are disabled.

INSTRUCTION IM 0, IM 1 and IM 2 should never be used in a TPA program.

INSTRUCTION DI should only be used when it is followed by an EI shortly afterwards and when there are no system calls between the DI and the EI.

#### 7.4. Terminating TPA program and return to Operating System

There are 3 basic methods of terminating a TPA program and returning control back to the operating system as follows:

1. A 'RET' instruction if the CCP Loader stack is used. Thus a TPA program could consist of only two instructions:

```
0100    NOP      ; (necessary to avoid conflict with RSK)
0101    RET
0102    END
```

2. A 'JP 0' instruction passes control to the WARM BOOT in Page Zero.
3. The BDOS function 0 SYSTEM RESET terminates the calling program and performs a WARM BOOT.

Each of the above 3 methods leads to a BIOS WARM BOOT.

In addition to the above basic methods, there are 3 further methods of terminating a program which controls the following activity.

1. The BDOS function 59 LOAD OVERLAY can be used to load a new program, terminate the calling program and pass control to the new program (without initializing page zero).
2. The BDOS function 47 CHAIN TO PROGRAM terminates the calling program and instructs the CCP to use the command line passed from this calling program (terminated with 00H). The command line may contain a built-in function, or program name to load a new program.
3. The calling program can create a file called \$\$\$SUB containing commands lines in fixed record lengths of 128 bytes. When the program terminates using any of the previous 3 basic methods, BDOS instructs the CCP to read the command lines from this file (in reverse order). Each command line, as in the CHAIN TO PROGRAM function, can contain a built in function, or program name to load a new program.

Additional program control can be obtain with the use of the Program Return Code through BDOS function 108. For example, CCP uses this to test for an unsuccessful return when the next command line begins with the conditional ':'. If the value is in the range FF00H-FFFDH or FFFFH, then the conditional command line is not executed.

## 7.5. Operating system interface BDOS, BIOS and HARDWARE

A TPA program has access to the operating system and system data through a number of locations as follows:

0001H-0002H	Address of the BIOS offset 3 bytes
0005H	Entry to BDOS (and any RSX's) functions
0050H	Drive from which TPA loaded

The following BDOS functions provide access to more system data:

12	Returns version number
25	Returns current default disk
49	Get system control block
50	Direct bios calls
107	Return serial number

The BDOS function 49 provides access to the System Control Block. It does not directly provide a pointer to the area, but this is easy to obtain as the word returned from offset 003AH contains the start address of the System Control Block. Either through function 49, or by using this address, the data contained in the 100 byte System Control Block can be accessed. This SCB contains a wealth of information about the system and is described in detail later in this chapter.

BDOS function 50 provides a restricted access to the BIOS functions. Included in the BIOS is the BIOS function 30 which is provided for access to special features of the customized BIOS. This may include access to hardware functions.



## 7.6. BDOS calling conventions

Access to the BDOS functions are made by a call to BDOS through the Page Zero jump vector at 0005H with the function number passed in register C. Apart from any intervention by RSXs, not all the implemented function numbers are strictly BDOS functions. Broadly the function numbers are allocated as follows:

- 0 thru 63 - BDOS functions
- 64 thru 95 - NDOS calls (Network Disk Operating System)
- 96 thru 127 - BDOS functions
- 128 thru 255 - XDOS functions (Extended DOS provided in MP/M)

Not all of the BDOS functions 0-63 are processed by the BDOS, the functions 59 and 60 are intercepted by RSXs, and functions 47 and 50 are routed by the BDOS for processing by the CCP program and by the BIOS functions respectively.

All calls to BDOS functions use a standard convention. On entry to the BDOS, register C contains the BDOS function number, and the register pair DE is used for any data that is required. This may be a byte or word value, or an address of a block of data. This block of data may be partially or completely filled with information and it may be modified, updated, or filled by the BDOS function.

BDOS functions return single byte values in register A, and double byte or word values in the register pair HL. In addition, BDOS functions return with register A = register L and register B = register H.

If a BDOS function is not supported then the return code depends on the function number:

- BDOS functions return HL = 00FFH (CP/M 2.2 returns 0000H)
- NDOS functions return HL = 00FFH (NDOS specifies FFFFH)
- XDOS functions return HL = 0000H (MP/M BDOS returns 0000H)

Note that some editions of the Digital Research CP/M Plus manual incorrectly state that the the return code should be FFFFH and not 00FFH as shown above.

XDOS functions usually return, in register A, a return code of FFH as indicating 'false' (failure to perform the function), and 00H to indicate that the function was properly performed, unless the function returns a specific value.

The following example illustrates a simple TPA program that uses BDOS calls. It reads and displays characters until a carriage return or line feed is encountered.

```

bdos    equ 0005h    ; BDOS entry point
conin   equ 1        ; BDOS console input
conout  equ 2        ; BDOS console output
cr      equ 0dh      ; <RETURN>
lf      equ 0ah      ; <LINE FEED>

cseg

0000 nextc:  ld  c,conin
0002         call bdos    ; Return character in A
0005         cp   cr      ; <RETURN> ?
0007         jr   z,eol   ; -yes-
0009         cp   lf      ; <LINE FEED> ?
000B         jr   z,eol   ; -yes-
000D         ld   e,a
000E         ld   c,conout
0011         call bdos    ; Echo character to screen
0014         jr   nextc   ; Loop for next character
0016 eol:    ld   e,cr
0018         ld   c,conout
001A         call bdos    ; Send <RETURN> on termination
001D         ret         ; Terminate program
001E         end

```

#### 7.6.1 Tools used in Programming Examples - M80, LINK and ZSID

As (almost) all CP/M Plus microcomputers use a Z80 processor, the examples in this chapter are provided in Zilog Z80 mnemonics. The assembler recommended is the Microsoft M80, and the linker is Digital Research's LINK. The Microsoft linker could be used, but the Digital Research linker is easier to use and contains extra features. The CP/M User group (UK) has a good public-domain Zilog assembler called Z80ASMUK, but it does not produce relocatable code.

Debugging of programs written using Zilog Mnemonics are difficult with the standard CP/M Plus debug utility SID; Firstly, the mnemonics are different but, more importantly, SID does not understand the instructions and registers that were added to the Z80 over and above the 8080. Digital Research, in their only acknowledgement to the Zilog Z80 mnemonics, produce a Z80 debugger called ZSID. Even though most computers use the Z80 processor, Digital Research does not supply an assembler with Z80 mnemonics. ZSID contains all the SID features but with the Z80 mnemonics and registers instead of the Intel 8080.

Unfortunately Digital Research do not include ZSID with the CP/M Plus release. The standard ZSID is produced for CP/M 2.2, but like the other 2.2 debuggers DDT and SID, they are not fully compatible with CP/M Plus firstly because they do not contain a 'W'rite facility, nor do they understand PRL bit maps. Under CP/M Plus, a modified version of SID is provided which has been adapted like the MP/M version to overcome these limitations and add extra features.

The CP/M 2.2 ZSID is, however, quite usable under CP/M Plus. Better still is to obtain and patch the MP/M version of ZSID (V2.5), assuming you can obtain the MP/M version. The patches are as follows (using CP/M Plus SID):

```

A>SID ZSID.PRL
&si2be
12EE C2 CA
12EF 32 .
&si296
1296 23 3E
1297 23 07
1298 7E 07
1299 07 07
129A 07 07
129B 07 4F
129C F6 .
&si2B1
12B1 19 69
12B2 7E 26
12B3 87 00
12B4 C6 36
12B5 30 C3
12B6 6F 23
12B7 60 00
12B8 36 .
&szsid.pr1
0062h record(s) written.
&q0

```

Note: If a different restart number is required other than the usual RST 7, then change the byte at 1297 to the restart number required (1 thru 7 only).

Digital Research have not produced a full set of manuals for all the version of ZSID and SID. The only full manual is for the CP/M 2.2 SID. For the CP/M 2.2 ZSID Digital Research produced a summary command pocket book. For the MP/M versions, Digital Research provided some notes on the differences, and for the CP/M Plus version of ZSID, the CP/M manual summarized the commands.

The ZSID (V2.5) and SID (V2.4) enhancements over the 2.2 versions under the MP/M Operating System are:

The Assemble (A) command - modified to include specification of relative address operands and update of the bitmap of a page relocatable file. Relative address is preceded by an '\*'.  
 .

The Bitmap (B) command - added to enable a page-relocatable bitmap to be updated. Bs,0 clears bit associated with address 's', and Bs,1 sets the associated bit.

The Normalize (N) command - relocates a page relocatable file loaded into memory by the debugger.

The Value (V) command - recalls the statistics printed when a file is loaded, and the command Vs calculates the number of sectors to write from 0100h upto the address s.

The Write Disk (Wn) command - writes a program image back to disk where 'n' is the number of sectors. The command I must be used to specify the filename.

The SID (V3.0) enhancements over the 2.2 versions under the CP/M Plus Operating System are:

The Load for Execution (E) command - load program and/or symbol table ready for execution.

The Read Code/Symbols (R) command - modified to include the file specification instead of requiring the prior use of the I command.

The Value (V) command - recalls the statistics printed when a file is loaded.

The Write Disk (W) command - writes to the file specification included a contiguous program image. The start and finish address may be added to the command.

## 7.7. BDOS character functions

CP/M Plus BDOS supports 5 logical character devices as follows:

1. Console input
2. Console output
3. List output
4. Auxiliary input
5. Auxiliary output

CP/M Plus allows up to 12 physical devices. The implementors of the CP/M Plus on each computer will determine the number of physical devices actually available.

Each of the 5 logical character devices may be assigned to one or more of the available physical devices, provided the input/output characteristic of the physical device matches the required logical device.

For example, if the computer provides both a parallel printer port connected to a fast matrix printer and a serial RS232 port connected to a serial Daisy Wheel printer, then the List device may be assigned to either of these ports, or even to both at the same time.

Any CP/M Plus system will support 5 logical devices, but the type and number of physical devices is specific to the particular hardware configuration. The use of logical devices enables application programs to access the keyboard, screen and printer through a standard logical device definition, rather than through a hardware-specific physical device definition which would require the application program to be configured for each hardware implementation.

The assignment of the 5 logical devices is provided through the appropriate redirection bits held in the BDOS System Control Block. Each bit corresponds to each of the 12 possible physical devices.

The description of the physical character devices is harder to access. In most cases the DEVICE utility provides all the likely information and access. This utility obtains the information from a table contained in the BIOS which provides the following essential information on each physical device:

1. Name for physical device
2. Whether an input, or output, or input/output device
3. If it has a serial RS232 interface
4. The baud rate of the serial RS232 interface
5. If X-ON/X-OFF handshaking is implemented

The character functions are split into 3 groups:

- 1 - Single character I/O
- 2 - String character I/O
- 3 - Direct console I/O

## 7.7.1 Single character I/O

The BDOS functions to send or receive a character to the logical devices are:

	CONS-IN	CONS-OUT	AUX-IN	AUX-OUT	LIST-OUT
status	11	-	7	8	-
transfer	1	2	3	4	5

Standard BDOS character functions

The BDOS parameters are:

status: Returned in register A 00H for not ready  
FFH if ready  
except function 11 which returns 01H if ready

transfer: Character to send is placed in register E  
Character to receive is returned in register A

The status returned by BDOS Function 11 can be altered by changing the Console Mode bit 0. If this is set to a 1, then the status is only returned TRUE when a CTRL-C is entered. This is used by PIP, for example, to abort copying from one file to another.

## 7.7.2 Character string I/O

In addition to the single character transfer, BDOS provides string functions for both the console and the list device:

	CONS-IN	CONS-OUT	AUX-IN	AUX-OUT	LIST-OUT
transfer	10	9 and 111	-	-	112

Standard BDOS character string functions

BDOS FUNCTION 10 - Read Console Buffer	
Register DE = Console buffer address	
Console buffer	
Byte 0	- Maximum number of characters which can be held in this buffer. The total length of the buffer is the value of this byte + 2.
Byte 1	- The number of characters placed in buffer
Byte 2	- Start of character string buffer
On entry the console buffer must be initialised before the BDOS function is called, as follows:	
Byte 0	- Maximum length no greater than 126 chars.
On return BDOS updates the console buffer as follows:	
Byte 1	- the length of the string returned.
Byte 2	- character string returned.
No terminator is added to the string.	

BDOS FUNCTION 10 - Read Initialised Console Buffer	
Register DE = 0000H (console buffer in current DMA)	
Current DMA address configured as console buffer	
Byte 0	- Maximum number of characters which can be held in this buffer. The total length of the buffer is the value of this byte + 2.
Byte 1	- The number of characters placed in buffer
Byte 2	- Start of character string. On entry a null (00H) terminator must follow the string.
On entry the console buffer must be initialised before the BDOS function is called, as follows:	
Byte 0	- Maximum length no greater than 126 chars.
Byte 2	- Initial character string terminated with a null byte (00H)
On return BDOS updates the console buffer as follows:	
Byte 1	- the length of the string returned.
Byte 2	- character string after editing.
No terminator is added to the string.	

BDOS FUNCTION 9 - Print string to logical device CONOUT: Register DE = String address
Starting at the string address, BDOS sends each consecutive character to the console output until the delimiter character is reached. The default delimiter is the dollar symbol \$, but this can be changed to any other character using BDOS function 110.
BDOS FUNCTION 110 - Get/Set Output Delimiter Register DE = FFFFH (GET), or output delimiter (SET)
Set output delimiter can change the BDOS FUNCTION 9 delimiter from the default of 'S' to any other byte value. The Get option enables a program to interrogate the current delimiter symbol.
BDOS FUNCTION 111 - Print block to logical device CONOUT: Register DE = CCB address
The character control block CCB contains: Bytes 0-1 - The 16 bit address of the character string Bytes 2-3 - The 16 bit length of the character string
BDOS FUNCTION 112 - List block to logical device LST: Register DE = CCB address
The character control block CCB contains: Bytes 0-1 - The 16 bit address of the character string Bytes 2-3 - The 16 bit length of the character string



## 7.7.3 Direct console I/O

BDOS supports a direct console input and output which provides a method of accessing the console logical devices without any character substitution and without any system control characters. Programs like Wordstar obtain the total control of console input by the use of direct console I/O. (Wordstar actually uses direct BIOS calls, but CP/M Plus BDOS converts these into direct console I/O function calls.)

Direct console input should not be mixed with any other console input or console output except for direct console output. Under CP/M 2.2, mixing direct console input with normal console i/o frequently leads to characters being misplaced in the normal console lookahead buffer. Under CP/M Plus this confusion has been reduced, but it is still possible for the BDOS incorrectly to access a character entered at the console which is intended for direct console input. Thus, for safety, it is best to use direct console I/O only when direct console input is used. Direct console output can be freely mixed.

+-----+   BDOS FUNCTION    6 - Direct console I/O   Register E    - Defined below 	
FFH	Return Console character if character ready,                      or 'Not Ready' input status of 00h if no                      character ready.
FEH	Return Console Input character status (00h                      if 'Not Ready', FFH if 'Ready')
FDH	Returns Console character. If no character                      'Ready' then waits until character is ready.
00H-FCH	Sends character in register E to console.
+-----+	

## 7.7.4 Character substitution and control

The BDOS console functions 1, 2, 9, 10, and 111 substitute the TAB character (CTRL-I) with blanks to align on columns of 8 characters.

The BDOS console functions 1, 2, 9 and 111 intercept any console input characters for the following special functions (unless the character mode has been changed).

- CTRL-S - 'Stop Scroll' Suspend console output until CTRL-Q received. (CTRL-P and CTRL-C may also be input)
- CTRL-Q - 'Start Scroll' Enables console output to continue following CTRL-S.
- CTRL-P - 'Echo/No Echo' Toggle to enable or disable echoing of console output to list device.

BDOS FUNCTION 1 - Input from logical device CONIN:

Calling parameters

Register C = 01H

Returned parameters

Register A - 8 bit ASCII character

Additional functions

Tab characters (CTRL-I) are expanded to 8 char column.  
Graphic characters, carriage return, line feed and back-space are echoed to the logical device CONOUT:  
If printer echo is enabled, characters are also echoed to the logical device LST:

Additional functions unless disabled by Console Mode

Input from CONIN: is scanned for  
CTRL-Q & CTRL-S to stop/start scroll  
CTRL-P to toggle printer echo enable-disable

BDOS FUNCTION 2 - Output to logical device CONOUT:

Calling parameters

Register C = 02H  
Register E - 8 bit ASCII character

Additional functions

If printer echo is enabled, characters are also echoed to the logical device LST:

Additional functions unless disabled by Console Mode

Tab characters (CTRL-I) are expanded to 8 char column.  
Input from CONIN: is scanned for  
CTRL-Q & CTRL-S to stop/start scroll  
CTRL-P to toggle printer echo enable-disable

BDOS FUNCTION 3 - Input from logical device AUXIN:

Calling parameters

Register C = 03H

Returned parameters

Register A - 8 bit ASCII character

```

+-----+
| BDOS FUNCTION  4 - Output to logical device AUXOUT: |
+-----+
| Calling parameters |
|   Register C = 04H |
|   Register E - 8 bit ASCII character |
+-----+

```

```

+-----+
| BDOS FUNCTION  5 - Output to logical device LST: |
+-----+
| Calling parameters |
|   Register C = 05H |
|   Register E - 8 bit ASCII character |
+-----+

```

```

+-----+
| BDOS FUNCTION  6 - Direct I/O to devices CONIN: & CONOUT: |
+-----+
| Calling parameters |
|   Register C = 06H |
|   Register E = 0FFH - returns CONIN: input or status |
|                   = 0FEH - returns CONIN: status |
|                   = 0FDH - returns CONIN: input |
|                   - 8 bit ASCII character to CONOUT: |
| |
| Returned parameters - E = 0FFH |
|   Register A = 0 if no console character ready |
|                   - 8 bit ASCII character |
| |
| Returned parameters - E = 0FEH |
|   Register A = FFH if a character is ready for input |
|                   = 00H if no character ready |
| |
| Returned parameters - E = 0FDH |
|   Register A - 8 bit ASCII character |
| |
| Note: Direct Console Input should not be mixed with |
|       either normal console input, or console output |
+-----+

```

```

+-----+
| BDOS FUNCTION   7 - Input status of logical device AUXIN: |
+-----+
| Calling parameters |
|   Register C = 07H |
|
| Returned parameters |
|   Register A = FFH if a character is ready for input
|               = 00H if no character ready
+-----+

```

```

+-----+
| BDOS FUNCTION   8 - Output status of logical device AUXOUT: |
+-----+
| Calling parameters |
|   Register C = 08H |
|
| Returned parameters |
|   Register A = FFH if logical device is ready for output
|               = 00H if logical device is not ready
+-----+

```

```

+-----+
| BDOS FUNCTION   9 - Output string to logical device CONOUT: |
+-----+
| Calling parameters |
|   Register C = 02H |
|   Register DE - Address of ASCII string terminated
|                 with the current delimiter (default $)
|
| Additional functions |
|   If printer echo is enabled, characters are also echoed to
|   the logical device LST:
|
| Additional functions unless disabled by Console Mode |
|   Tab characters (CTRL-I) are expanded to 8 char column.
|   Input from CONIN: is scanned for
|       CTRL-Q & CTRL-S to stop/start scroll
|       CTRL-P to toggle printer echo enable-disable
|
| String Delimiter |
|   The default string delimiter is the dollar symbol $, but
|   it can be changed to any 8 bit value by BDOS function 110
|   Get/Set output delimiter.
+-----+

```

---

BDOS FUNCTION 10 - Input line from logical device CONIN:

---

## Calling parameters

Register C = 0AH

Register DE - Address of Console Buffer Block

= 0000H (Console buffer placed in DMA)

DMA ADDRESS - If DE = 0000H, DMA ADDRESS set to address  
 (BDOS 26) of Console Buffer Block containing an  
 initial character string terminated with a  
 binary zero.

## Returned parameters

Characters placed in Console Buffer Block when either a  
 CTRL-M (RETURN) or CTRL-J (LINE=FEED) is encountered.

## Format of Console Buffer Block

CBB: db mx ; maximum character capacity

CBB\_NC: db 0 ; returned number of characters

CBB\_C: ds mx ; returned characters

## Additional functions - DE = 0000H

The Console Buffer Block is initialised with a character  
 string starting at CBB\_C and terminated with a binary  
 00H. This string is sent as output to the Logical Device  
 CONOUT: and is used as the initial character string to be  
 returned or edited.

## Additional functions

The read console buffer function reads a line of edited  
 input characters and returns when either a CTRL-M  
 (CARRIAGE-RETURN) or CTRL-J (LINE-FEED) is entered. The  
 returned character string excludes the line terminator  
 (CTRL-M or CTRL-J). If the characters entered exceed  
 the capacity of the buffer, then these are lost and a  
 BELL (CTRL-G) is sent to the console output.

If printer echo is enabled, characters are also echoed to  
 the logical device LST:

---

---

BDOS FUNCTION 10 - Edit Control Characters

---

## Edit Control Characters

DEL Deletes character to left of cursor.  
 CTRL-A Cursor moved LEFT one character.  
 CTRL-B Cursor moved to start at line, or to the end of line if already at the start of the line.  
 CTRL-C REBOOTS CP/M if entered as the first character in the line.  
 CTRL-E Insert NEWLINE into display.  
 CTRL-F Cursor moved RIGHT one character.  
 CTRL-G Delete character under cursor.  
 CTRL-H (BACKSPACE) Deletes character to left of cursor.  
 CTRL-J (LINE-FEED) Terminates character input. The cursor does not have to be positioned at the line end.  
 CTRL-K Deletes character under cursor and all characters to right of cursor.  
 CTRL-M (RETURN) Terminates character input. The cursor does not have to be positioned at the line end.  
 CTRL-P Enables/Disables printer echo toggle.  
 CTRL-R Echoes the characters to the left of the cursor on to a new line.  
 CTRL-U Moves all characters to the left of the cursor to the previous line buffer, deletes current line, and moves cursor to start of new line ready for input.  
 CTRL-W Copies all characters in the 'previous line' buffer to the current line if the cursor is at the start of the line, otherwise moves cursor to end of line.  
 CTRL-X Deletes all characters to the left of the cursor.

Edit Control changed by Console Mode

CTRL-C Reboot may be disabled

---

## BDOS FUNCTION 11 - Input status of logical device CONIN:

Calling parameters  
Register C = 0BH

Returned parameters  
Register A = 01H if a character is ready for input  
          = 00H if no character ready

Alternative returned parameters if enabled by Console Mode  
Register A = 01H if CTRL-C entered  
          = 00H if no CTRL-C character entered

## Important note:

Usually BDOS returns a value of FFH as a 'READY' or 'TRUE' status, but this function returns 01H.

Entering a CTRL-C usually deletes all characters in the CONIN: buffer waiting to be read.

## BDOS FUNCTION 109 - Get/Set Console Mode

Calling parameters  
Register C = 6DH  
Register DE - Console Mode (SET)  
              = FFFFH (GET)

Returned parameters (DE = FFFFH)  
Register HL = Console Mode (GET)

## Console Mode Bits (all bits set to 0 by CCP)

Bit No.		Used by Functions
0 = 0	Status refers to any character	11
= 1	Status refers to CTRL-C character	
1 = 0	CTRL-S & CTRL-Q characters filtered from CONIN: to stop/start scroll	1, 2, 9 & 111
= 1	CTRL-S & CTRL-Q not filtered	
2 = 0	CTRL-I character to CONOUT: replaced expanded to 8 character columns, and supports printer echo (CTRL-P) control	2, 9 & 111
= 1	CTRL-I not expanded, no printer echo	
3 = 0	CTRL-C character causes WARM START after CTRL-S and if first character in console buffer.	2, 9, 10 & 111
= 1	CTRL-C is not intercepted	
8 & 9	reserved for RSX console status control	

## BDOS FUNCTION 110 - Get/Set output delimiter

## Calling parameters - GET

Register C = 6EH

Register DE = FFFFH (GET)

## Calling parameters - SET

Register C = 6EH

Register E = Delimiter byte (00H-FFH)

Register D = 00H

## Returned parameters - GET (DE = FFFFH)

Register A = Current output delimiter

The output delimiter is used by BDOS function 9 to determine the terminator of the passed string. It is initially set by CCP to the dollar sign '\$'. This function enables the delimiter to be changed, when, for example, the string to be output to the CONSOLE contains dollars.

## BDOS FUNCTION 111 - Output string to logical device CONOUT:

## Calling parameters

Register C = 6FH

Register DE - Address of Character Control Block

## Character Control Block (CCB)

## Format of Character Control Block (CCB)

```
CCB:  dw  blk      ; Address of character string
      dw  l_blk    ; Length of character string
```

```
blk:  db  '.....' ; Character string
```

```
l_blk equ $-blk    ; Length of character string
```

## Additional functions

If printer echo is enabled, characters are also echoed to the logical device LST:

## Additional functions unless disabled by Console Mode

Tab characters (CTRL-I) are expanded to 8 char column.

Input from CONIN: is scanned for

CTRL-Q & CTRL-S to stop/start scroll

CTRL-P to toggle printer echo enable-disable



---

BDOS FUNCTION 112 - Output string to logical device LST:

---

## Calling parameters

Register C = 70H

Register DE - Address of Character Control Block

## Character Control Block (CCB)

## Format of Character Control Block (CCB)

CCB: dw blk ; Address of character string

dw l\_blk ; Length of character string

blk: db '.....' ; Character string

l\_blk equ \$-blk ; Length of character string

## Additional functions

If printer echo is enabled, characters are also echoed to the logical device LST:

## Additional functions unless disabled by Console Mode

Tab characters (CTRL-I) are expanded to 8 char column.

Input from CONIN: is scanned for

CTRL-Q &amp; CTRL-S to stop/start scroll

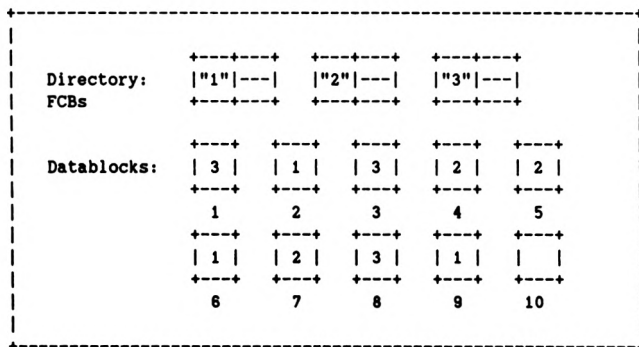
CTRL-P to toggle printer echo enable-disable

## 7.8. BDOS disk functions

The BDOS disk system organises the files of data for storage on a disk system. The files are described by entries in a directory which are also stored on the disk system. These directory entries provide both the filename and the location of the data associated with that file.

The disk system is organised into datablocks, within which the data is held sequentially. The datablocks are accessed randomly, in that there is no direct link or connection between any two datablocks (MSDOS, for example, does provide a link between datablocks). Datablocks are allocated to a file as required, but only de-allocated when a file is erased, or when the end of file pointer is reduced.

The file entry in the directory is the 'Directory File Control Block' or Directory FCB. This FCB includes the numbers of up to 16 datablock numbers allocated to the FCB. The disk may be considered as a set of boxes which are allocated to a directory of FCBs as shown in the following diagram, where the number in the datablocks refer to the FCB to which that datablock is allocated. An empty box is unallocated.



Example of random allocation of datablocks to FCBs

The allocation of datablocks is deduced by scanning all the directory FCBs to see which datablocks are attached to the FCBs, thereby an allocation table can be constructed showing which datablocks are allocated and which are free. This allocation table is the fundamental part of the CP/M BDOS operation.

The File Control Blocks or FCBs, which are passed between programs and the BDOS, are very similar to the directory FCB. The (BDOS) FCB and the Directory FCB share much of the same information; However, the FCBs are modified slightly from the Directory FCB and include extended fields, as shown in the following table:

```

1. Directory File Control Block - Datablocks < 257
-----
|
| +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+
| |ur|f1|f2|f3|f4|f5|f6|f7|f8|t1|t2|t3|ex|s1|s2|rc|
| +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+
| 00 01 02 03 04 05 06 07 08 09 10 11 12 13 13 15
|
| +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+
| |d0|d1|d2|d3|d4|d5|d6|d7|d8|d9|da|db|dc|dd|de|df|
| +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+
| 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
|
|-----
|
|
|-----
2. Directory File Control Block - Datablocks > 256
-----
|
| +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+
| |ur|f1|f2|f3|f4|f5|f6|f7|f8|t1|t2|t3|ex|s1|s2|rc|
| +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+
| 00 01 02 03 04 05 06 07 08 09 10 11 12 13 13 15
|
| +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+
| | d0 | d1 | d2 | d3 | d4 | d5 | d6 | d7 |
| +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+
| 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
|
|-----
|
|-----
3. BDOS FCB
-----
|
| +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+
| |dr|f1|f2|f3|f4|f5|f6|f7|f8|t1|t2|t3|ex|s1|s2|rc|
| +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+
| 00 01 02 03 04 05 06 07 08 09 10 11 12 13 13 15
|
| +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+
| |d0 ..... dn|cr|r0|r1|r2|cr|r0|r1|r2|
| +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+ +---+
| 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
|
|-----

```

FCB Items	
-----	
dr	Drive code (0,1-16) (0 = current default, 1 = A etc)
f1..f8	7 bit filename, 8th bit used as attributes
t1..t3	7 bit filetype, 8th bit used as attributes
ex	Bits 0-4 contain low bits of Extent number (0-31)
s1	Reserved for system use whilst file in use
s2	Bits 0-5 contain extended Extent number
	Bits 6-7 reserved for system use whilst file in use
rc	Record count in this extent (0-128)
d0..dn	Filled in by BDOS with datablock allocation
cr	Current record for next sequential read or write
	Normally set to zero before file is opened, and
	BDOS increments by one for each sequential access.
	When CR=128, BDOS opens next extent and resets CR to zero.
r0..r2	Random record number (0-3FFFFH) set by user prior to any random read or write. BDOS does not increment
Directory File Control Block Items	
-----	
ur	Bits 4-7 Define Directory Entry
	0000B -> File Control Block
	0001B -> XFCB containing password
	0010B -> Directory label or date and time stamp
	1110B -> Erased or unused File Control Block
	Bits 0-3 Define File Control Block User number
f1..f8	7 bit filename, 8th bit used as attributes
t1..t3	7 bit filetype, 8th bit used as attributes
ex	Bits 0-4 contain low bits of highest Extent number
s1	Optional Byte Count for last record in highest extent
s2	Bits 0-5 contain extended highest Extent number
rc	Record count in highest extent (0-128)
d0..d7	Filled in by BDOS with 16 bit datablock numbers
d0..df	Filled in by BDOS with 8 bit datablock numbers

The directory FCB contains the user number (0-15) in the first byte. On any BDOS function using the FCB, the current user number is compared with the directory FCB user name as part of any matching of file names. In order to access files with different user numbers, the BDOS FUNCTION 32 must be utilised to change the user number before each file access. There is an exception to this when accessing SYStem files on user 0.

#### LOGICAL, PHYSICAL, CURRENT, AND DEFAULT DRIVES

The CP/M BDOS disk system provides for up to 16 logical disk devices. A logical disk device may be assigned to a physical drive such as a floppy drive, or one physical disc such as a winchester drive may be partitioned into one or more logical disk devices. The physical disk device need not be a floppy or winchester disc, it could be 'silicon ram' memory configured to emulate a disc, or a tape device which may be used as a winchester backup.

The CP/M BDOS logical disks are assigned logical device names 'A:', 'B:' etc. up to 'P:'. Associated with each name is a drive number which may be either 1 through 16 or 0 through 15. The BDOS maintains a 'current default' disk which is initially set by the CCP to the 'Default' disk as displayed by the CCP prompt. Selection of this 'current default' disc is made by specifying the drive number 0, and where this is used the number for drive 'A:' is 1. BDOS functions which do not permit the reference to the 'current default' drive, use the number 0 for drive 'A:'. This can cause confusion if the incorrect drive specification is used. In general, any BDOS function involving an FCB includes the default specification, whilst any BDOS function involving the drive explicitly excludes the default specification. Exceptions are found at zero page byte 50 which includes the default, the drive search chain and temporary drive held in the SCB (System control block).

The CP/M Plus BDOS disk functions comprise the following groups:

1. Drive functions
2. Sequential file access
3. Random access
4. Random access using sequential disk functions
5. Directory access
6. Special functions

#### BDOS DISK FUNCTIONS

-----

## 7.8.1. BDOS Drive functions

SELECT DISK and RETURN CURRENT DISK

The BDOS FUNCTION 14 - SELECT DISK and BDOS FUNCTION 25 - RETURN CURRENT DISK provide access to the current default disk which is initially set up by the CCP. The BDOS FUNCTION 14 changes the current default disk and if the new drive is not currently logged-in, it is selected. BDOS FUNCTION 25 returns the currently selected (default) disc drive.

CURRENT USER NUMBER

The BDOS FUNCTION 32 - SET/GET USER NUMBER can either return the current user number, or change the user number. The current user number is initially set by the CCP to the default user number which is displayed at the CCP prompt if it is not zero. The current user number is appended to the FCB filename before scanning the directory for a matching FCB, except that the BDOS function 15 to OPEN an FCB will also search user 0 for a matching filename. Consequently, if the subsequent user 0 matches an FCB with the SYSTEM attribute t2' set, the FCB is opened with the attribute f8' set so that, on all subsequent access to the opened FCB, the filename is searched on user 0.

RESET DISK SYSTEM and RESET DRIVE

All drives are initially in the reset state when the CP/M Plus BDOS is first loaded. As the drives are selected, the drives are placed in a logged-in state. Resetting a drive increases the level of file security by ensuring that before the next access to the drive, the directory and drive characteristics are re-read. This will minimise any difficulties resulting from the change of diskettes. Before resetting any drive, all files should be closed.

BDOS FUNCTION 13 - RESET DISK SYSTEM resets all logged-in drives and resets any write-protect status. BDOS FUNCTION 37 - RESET DRIVE enables one or more of the 16 logical drives to be reset.

RETURN LOGIN VECTOR, WRITE PROTECT DISK, & GET READ-ONLY VECTOR

The CP/M BDOS maintains a 16 bit login vector and a 16 bit read-only vector in which each bit records the status of each of the 16 logical drives. Initially both vectors are set to zero, then as drives are selected and reset the login vector bits are set and reset accordingly.

These two 16 bit vectors behave in an unexpected manner through historical CP/M defects. Early versions of CP/M only provided a maximum of 4 drives (A through D) and not the current maximum of 16 drives. Thus the LOGIN vector occupied only one byte and so, naturally, bit 0 was chosen to represent drive A. With a 16 bit vector it is logical to choose bit 15 to represent drive A, but to maintain compatibility, bit 15 represents drive P. The READ-ONLY vector was introduced as a protection against the accidental write to a diskette when the diskette had been changed after the file was opened. CP/M maintains a directory checksum for each removeable drive, and before each WRITE, the BDOS checks this checksum. If the checksum is incorrect, then earlier versions of BDOS used to marks the diskette as read-only to stop such a write. Previous versions of CP/M had no method of actually testing if the diskette itself was write-protected. Consequently the READ-ONLY vector DOES NOT show the diskette

protect status. Thankfully, CP/M Plus can detect a write-protected diskette, but passes this information in the form of errors on any attempt to write to a write-protected disc.

Because of the automatic disk re-selection provided under CP/M Plus, a directory checksum error should not lead to the read-only vector remaining set, hence the read-only bits can only be set by the BDOS FUNCTION 28 - WRITE PROTECT DISK for each drive. It is not set by the physical write-protect tab which can be placed on the floppy disc media. The read-only status is reset by BDOS FUNCTION 13, FUNCTION 37, or an automatic disc re-selection.

BDOS FUNCTION 24 - RETURN LOGIN VECTOR returns the 16 bit login vector, and BDOS FUNCTION 29 - GET READ-ONLY VECTOR returns the 16 bit read-only vector.

#### ACCESS DRIVE and FREE DRIVE

Originally, CP/M Plus was designed to operate with background processes, but this was dropped on the 8 bit version, whereas it was maintained in the 16 bit version (which was never released), and supported in the DOS Plus operating system. To support file sharing, it is necessary to prohibit the resetting of drives by one process and thereby interfering with the activity of a second process. BDOS FUNCTION 38 - ACCESS DRIVE sets a flag to prohibit another process from resetting one or more drives, and the BDOS FUNCTION 39 - FREE DRIVE resets this flag. Under CP/M Plus these functions do not perform any action.

BDOS FUNCTION 13 - RESET disk system

Calling parameters  
Register C = 0DH

Returned parameters  
Default disk is set to drive A:  
DMA address is set to 0080H

Additional functions  
Any pending blocking/deblocking data is discarded.  
Any newly allocated datablocks on unclosed file extents  
are lost.  
Each bit in the LOG-IN vector is set to zero.  
Each bit in the READ-ONLY vector is set to zero.

BDOS FUNCTION 14 - Select disk

Calling parameters  
Register C = 0EH  
Register E - Selected disk number  
Disk number = 0,1 thru 15 for drive A:, B:, thru P:

Returned parameters  
Register A - Error flag  
Register H - Physical error

Additional functions if the selected drive is reset.  
The default disk is 'logged-in'.  
The drive allocation vector is scanned and checked.  
The drive directory check-sum is calculated.  
Any drive directory hash table is created.  
Any error in 'log-in' produces error return.

Error returns  
Error flag = 00H: Successful operation  
= FFH: Physical error in extended error mode

Physical Error returns (extended disc error mode only)  
= 01H: Disk I/O error  
= 04H: Invalid drive (drive does not exist)



BDOS FUNCTION 24 - Return login vector

Calling parameters  
Register C = 18H

Returned parameters  
Register HL = 16 bit login vector

Login Vector  
Bit 0, or the least significant bit of the 16 bit vector is the login bit for drive A:. A value of 1 indicates the drive is active.  
The Bits 0 to 15 correspond to drives A: to P:

BDOS FUNCTION 25 - Return current disk

Calling parameters  
Register C = 19H

Returned parameters  
Register A = Current disk

Current disk  
The disk number ranges from 0 to 15 corresponding to drives A: through P:. It should not be assumed that the current disk has been logged-in or that it actually exists.

BDOS FUNCTION 28 - Write Protect disk

Calling parameters  
Register C = 1CH

Earlier versions of CP/M did not provide for any flags to indicate if the floppy diskette had the write protect tab attached. Instead a disc was flagged as 'Write Protect' if the diskette produced a directory checksum error.  
This function was provided to set the 'Read Only' vector.  
Under CP/M Plus, this feature is unnecessary.

---

 BDOS FUNCTION 29 - Get Read-Only vector
 

---

Calling parameters  
 Register C = 1DH

Returned parameters  
 Register HL = 16 bit read-only vector

Read-only Vector  
 Bit 0, or the least significant bit of the 16 bit vector  
 is the Read-Only bit for drive A:.  
 The Bits 0 to 15 correspond to drives A: to P:

Note:  
 Under CP/M Plus, these bits are only likely to be set  
 following a BDOS FUNCTION 28 call.

---



---

 BDOS FUNCTION 32 - SET/GET USER NUMBER
 

---

Calling parameters  
 Register C = 20H  
 Register E = 0FFH - GET USER NUMBER  
           or E - user number (0-15) - SET USER NUMBER

Returned parameters - GET USER NUMBER  
 Register A - user number

Returned parameters - SET USER NUMBER  
 Register A - 0  
 SCB offset 44H - Set to bits 0-3 of E

Note:  
 Under CP/M 2.2 it was possible to set the user number  
 in the range 0-31. Under MP/M and CP/M Plus the range  
 is restricted to 0-15. The CCP default user number, even  
 under CP/M 2.2, is also limited to the range 0-15.

---

BDOS FUNCTION 37 - RESET drive

Calling parameters

Register C = 25H

Register DE = 16 bit drive vector

Drive vector

The Bits 0 to 15 correspond to drives A: to P:

Bit 0, or the least significant bit of the 16 bit vector is the drive bit for drive A:. A value of 1 indicates that the drive is to be reset.

Additional functions for each drive to be reset

Any pending blocking/deblocking data is discarded.

Any newly allocated datablocks on unclosed file extents are lost.

The corresponding bit in the LOG-IN vector is zeroed.

The corresponding bit in the READ-ONLY vector is zeroed.

Note:

Under the previous version 2.2 of CP/M, this function produced errors if this call was used to reset the current drive.

BDOS FUNCTION 38 - Access drive

Calling parameters

Register C = 26H

Register DE = 16 bit drive vector

Returned parameters

Register A - Error flag

Register H - Physical error

The format of the Drive vector is the same as Function 37

This function performs no action under CP/M Plus and returns a value of 00H in register A

BDOS FUNCTION 39 - Free drive

Calling parameters

Register C = 27H

Register DE = 16 bit drive vector

Returned parameters

Register A - Error flag

Register H - Physical error

The format of the Drive vector is the same as Function 37

This function performs no action under CP/M Plus and  
returns a value of 00H in register A

### 7.8.2. Sequential file access

The BDOS functions to access files sequentially are:

- 45 - Set BDOS error mode
- 152 - Parse Filename
- 22 - Make File
- 15 - Open File
- 26 - Set DMA transfer address
- 44 - Set Multi Sector count
- 20 - Read sequential
- 21 - Write sequential
- 16 - Close File
- 19 - Delete File

BDOS function 45, SET BDOS ERROR MODE, is optional. By changing the error mode, one can stop the program crashing out with the BDOS BAD SECTOR message (Because this message often caused excruciating pain when it appeared under earlier releases of CP/M, this message become synonymous with CP/M). Using this function enables a program to retain control whatever the BDOS disc error. Its use is highly recommended.

BDOS function 152 PARSE FILENAME is a very useful function taken from the MP/M XDOS function. A character string is passed to the function which returns a FCB initialised to the first filename found. It also returns the address of any filename delimiter for easy decoding of multiple filenames. Under CP/M 2.2, a lot of program code was required to decode and initialise an FCB correctly. This call saves a lot of program work and is highly recommended for the construction of any FCB.

Before a file can be accessed it must first be 'opened'. A file can be opened by creating a new file or by opening an existing file. In both cases the FCB must not only be initialised with the file name and drive, but also with the the EX (file extent) and the CR (current or next record in extent) fields. EX must be set to zero prior to opening, and CR should be set to zero ready for sequential access. CR may, however, be set to OFFH in the OPEN function which will then return the last record byte count in the FCB field CR. The BDOS function 152 initialises all the required fields in the FCB, except for CR, R0, R1, and R2. It is the usual practise to set CR, R0, R1 and R2 to zero after using this function 152. On opening an FCB, BDOS completes the fields S1, S2, RC and the 16 bytes of datablock. The BDOS also first sets S2 to zero on a call to 'open' a file.

Under earlier releases of CP/M, it was not essential to open the extent 0 of a file, but under both MP/M and CP/M Plus, the BDOS cannot perform all the functions properly if this is not done.

BDOS function 22 MAKE FILE creates a new directory entry for a file for the specified filename and extent. It also creates an extended directory entry if the disk label enables passwords and a password is supplied. If time and date stamping are operative, then the CREATE and UPDATE stamps are initialised if extent 0 is being created. If a password is to be assigned to the file, it is placed in the first 8 bytes of the current DMA buffer, the password mode byte is placed in the 9th byte of the DMA buffer, and the attribute f6' is set to a 1. F6' is the high bit (bit 7) of the 6th letter of the 8 letter file name. The password mode bits are set as follows (see also BDOS Function 102):

Bit 7 - Password READ protect (and write and delete)  
 Bit 6 - Password WRITE protect (and delete)  
 Bit 5 - Password DELETE protect

Unlike CP/M 2.2 and earlier releases, MAKE FILE will set an error condition if either the file already exists, or if the filename contains the wildcard character '?'. (Under CP/M 2.2 it was possible to create multiple entries in the directory with the same filename and extent). On a successful return, MAKE FILE initialises the FCB into the OPEN condition.

BDOS function 15 OPEN FILE must be used to 'OPEN' an FCB for any existing file before the FCB can be used in any BDOS function to transfer disc data and the BDOS functions CLOSE, SET RANDOM RECORD, LOCK and UNLOCK. OPEN FILE function opens an FCB for the specified filename and extent. If the file is password protected, then a password must be supplied by placing the password in the first 8 bytes of the current DMA buffer, or by the previous use of BDOS FUNCTION 106 to set a default password. If the password is not matched, then if the file is READ protected, the open fails, if the file is only WRITE password protected then the attribute bit f7' is set.

If the CR field of the passed FCB is set to 0FFH (instead of the usual 00H), this selects an additional feature of the OPEN FILE function to return in CR the value set by a previous call to Function 30 to set the Directory FCB field S1. This is usually used to set the count for the last byte in the last record. BDOS only maintains a pointer to the last record in a file, but this feature makes it possible to maintain an EOF pointer to the last byte in a file.

If the OPEN function cannot find a matching file name and extent for the current user number, and if the current user number is not zero, the directory is searched for a matching file name with user number zero. If a file is found it is opened only if the SYStem attribute t2' is set and then the attribute bit f8' is set on the FCB. (This user 0 search feature was not supported under CP/M 2.2, whilst MP/M also required attribute t1' to be set.) Such a file may only be used for READ access.

BDOS function 26 SET DMA TRANSFER ADDRESS is used to change the DMA address from the default value of 0080H set by the CCP when the program is loaded. The DMA buffer is used primarily for the data transfer, but it is also used for passing passwords and for initialising the read console buffer. However the DMA buffer used to pass parameters to the BDOS Function 47 Chain To Program is always the CCP default buffer of 0080H, because this function is processed by the CCP and not BDOS. The DMA buffer is usually 128 bytes in length, more for multi sector transfers, and less for password and console parameters. BDOS does not update the DMA address following a disc data transfer; thus the DMA address is usually set prior to each sequential data transfer. The alternative is to keep a fixed DMA transfer buffer and to copy the data to/from the actual data area.

BDOS function 44 SET MULTI SECTOR COUNT allows for the sequential transfer of more than one record of 128 bytes of data. The multi sector count is initialised to 1 by the CCP, and this function can change the count to any value between 1 and 128 records. Thus a total of 16k bytes may be transferred with one BDOS READ or WRITE call. Once the multi sector count is changed, it remains in effect for all further data transfers until it is reset. It is therefore safer to reset the count to 1 immediately after the data transfer to avoid mistaken multi sector transfers in other parts of the program. The multi sector count applies both to sequential and random data transfers.

BDOS function 20 READ SEQUENTIAL transfers the data for the file specified in the passed FCB using the current DMA address. The transfer starts from the record number specified in CR for the extent in EX and S2. The record number in CR must be in the range from 0 to 127, however if CR is set to 128, then BDOS increments the extent number, sets CR to zero and opens this extent before the read begins. After a successful read BDOS increments the CR to the value of the next record to read or to 128 for the record 0 of the next extent. The number of records read is determined by the current value of the multi sector count. If an attempt is made to read a non existing record, the number of records actually read is returned in register H with a value from 0 to 127.

BDOS function 21 WRITE SEQUENTIAL transfers the data for the file specified in the passed FCB using the current DMA address. The transfers starts with the record number specified in CR for the extent in EX and S2. The record number in CR must be in the range from 0 to 127, however if CR is set to 128, then BDOS closes the current extent, increments the extent number, sets CR to zero and opens this extent before the write begins. If the extent does not exist, it is created. After a successful WRITE, BDOS increments the CR to the value of the next record to write or to 128 for the record 0 of the next extent. The number of records written is determined by the current value of the multi sector count. If an error occurs in writing to a record, the number of records actually written is returned in register H with a value from 0 to 127. If the BDOS allocates a new datablock, it records this allocation in a temporary table (assuming the system implementation includes this). The permanent allocation table is not updated until an extent is closed. (See BDOS function 98 Free Blocks)

BDOS function 16, CLOSE FILE, must be used after completing the writing to a file to update the directory with the correct file allocation. The action of setting attribute f5' updates the directory, but leaves the file FCB in the 'opened' state. It is not essential to close a file which has only been read (BDOS only updates the directory if a file has been written to), but for compatibility with MP/M all files should be closed when access has been completed. The BDOS also transfers any temporary allocation of datablocks to its permanent allocation table. It is not possible to use the CLOSE FILE function to truncate a file (by changing the RC byte) as was the case in earlier versions of CP/M. A separate function is provided to do this.

BDOS function 19, DELETE FILE, removes all extents of any file which matches the filename and filetype in the FCB. The filename and filetype can contain the wildcard character '?'. When any directory entry is deleted, the directory FCB is marked with an 0E5H in the first byte and the datablocks allocated to that file are removed from the allocation table making them immediately available for allocation to other files. This release of datablocks was not generally available in competitive operating systems when CP/M was first developed, and even now some operating systems fail to release datablocks for re-use without running special utilities.

## 7.8.3. Random access

The BDOS functions to access files randomly are:

- 45 - Set BDOS error mode
- 152 - Parse Filename
- 22 - Make File
- 15 - Open File
- 26 - Set DMA transfer address
- 36 - Set Random Record
- 35 - Compute file size
- 44 - Set Multi Sector count
- 33 - Read random
- 34 - Write random
- 40 - Write random with zero fill
- 99 - Truncate File
- 16 - Close File
- 19 - Delete File

Many of these functions are also used for sequential access, and described in the preceeding paragraphs.

Random access differs from sequential access by the use of the random record number to specify the record to be accessed rather than the use of CR the current record, EX the extent, and S2 the extent overflow. Unlike sequential access, the random record number must be set before each access. In sequential access, the record number is automatically incremented after each access, but in random access, the record number is left unchanged.

Random access uses the 24 bit number in R0, R1 and R2 to specify the number of the 128 byte record to be accessed. Under CP/M Plus the random record number can range from 0 to 3FFFFH (32M bytes). Such a 32 M Byte file would have 2048 logical extents. This size of extents exceeds the space provided in a single byte and so requires more than one byte in the FCB to identify the extent. The Digital Research Documented FCB does not disclose how the extent number is held, the extent number is, however, calculated by concatenating bits 0-4 of the EX byte and bits 0-5 of S2 into an 11 bit extent number. In random access, the BDOS converts the 24 bit record number into this 11 bit extent number to find the correct FCB, and a 7 bit record number to determine the record in that extent to be accessed.

The random access is intended as a method of either reading or updating a file which was first created sequentially. It is possible to create a non-sequential file, that is a file which has 'holes' in it, but utilities such as PIP cannot properly access files with holes. There is an additional complication in creating files with holes because of the disc storage allocation technique used by CP/M. CP/M allocates disk space in units of datablock. The size of a datablock can be 1k, 2k, 4k etc. up to 16k. Thus when one record is written in a datablock, the remaining records in the datablock are undefined, and on reading there is no method of detecting such undefined records except through the EOF record pointer. The BDOS FUNCTION write with zero fill is provided to fill all the remaining records with zeroes, but there is no means of determining if a record was actually written, or just created because another record in the datablock was written. As it is very difficult, and time consuming, for the program to calculate if a write will result in additional records being created, programs should avoid true random files. A viable exception could though be made for ISAM (Indexed) files. A further disincentive to true random files is the potential for wasteful disk space



which in a worst case example could result in data occupying say 200 records (25k) actually occupying from 200k to 3,200k of disk space (depending on the datablock size) if each record happened to occupy a different datablock.

Files can be created sequentially with the random access functions with the use of the BDOS Function 35 COMPUTE FILE SIZE which actually returns the next record number beyond the current End Of File. Thus a new record can be added to the end of a file by using Function 35, thereby creating a sequential file.

Some of the random functions are identical to the functions used for sequential access including:

```
BDOS function 45 SET BDOS ERROR MODE
BDOS function 152 PARSE FILENAME
BDOS function 22 MAKE FILE
BDOS function 15 OPEN FILE
BDOS function 26 SET DMA TRANSFER ADDRESS
BDOS function 44 SET MULTI SECTOR COUNT
BDOS function 16 CLOSE FILE
BDOS function 19 DELETE FILE
```

Before a file can be accessed it must first be 'opened'. A file is opened either by creating a new file or by opening an existing file. In both cases the FCB must be initialised, not only with the file name and drive, but also the EX field. For Random Access the first extent (EX = 0) must always be opened before any access is made. This is necessary to ensure proper use of the Password and Date stamp options, and to provide compatibility with MP/M and the multi tasking versions of CP/M, even though it is not strictly essential. The first extent does not necessarily have to have any datablocks allocated, all that is required is an entry in the directory for extent 0.

When opening a file, it is good practice to set the CR, R0, R1, and R2 fields to zero as well as EX. CR may, however, be set to OFFH in the OPEN function to return the last record byte count as for the sequential open.

BDOS function 36 SET RANDOM RECORD is used to calculate the random record number from the current sequential record number (CR) in the extent defined by the fields EX and S2 in the FCB. This is a useful function for switching from sequential access to random access. The random record number is calculated by creating an 11 bit extent number with bits 0-4 taken from bits 0-4 of the EX field, and bits 5-10 taken from bits 0-5 of the S2 field. This 11 bit number is multiplied by 128 and added to the 8 bit number in CR to determine the 24 bit random record number (the high 5 bits are always zero). The function sets R0, R1 & R2 in the passed FCB.

BDOS function 35, COMPUTE FILE SIZE, calculates the record number of the End Of File plus one and returns this in R0, R1 & R2 of the passed FCB. The record number is calculated in the same way as for the BDOS function 36 except the directory is searched for the highest extent entry and the field RC is used instead of CR.

BDOS function 33 READ RANDOM transfers the data for the file specified in the passed FCB using the current DMA address. The transfers starts from the 24 bit record number specified in R0, R1 & R2. BDOS will calculate the extent number and open this extent if it differs from the values of the combined FCB fields EX and S2 supplied in the FCB, before the read begins. After a successful read BDOS updates the FCB fields EX, S2 and CR to correspond with

the random record number read, but unlike the sequential read, the record number is not incremented ready for the next sequential read. The number of records read is determined by the current value of the multi-sector count. If an attempt is made to read a non existing record, the number of records actually read is returned in register H with a value from 0 to 127.

BDOS function 34 WRITE RANDOM transfers the data for the file specified in the passed FCB using the current DMA address. The transfers starts from the 24 bit record number specified in R0, R1 & R2. BDOS will calculate the extent number and if it differs from the values of the passed FCB fields EX and S2, will first close the current extent, and open the new extent before the write begins. If the extent does not exist, it is created. After a successful write BDOS updates the FCB fields EX, S2 and CR to correspond with the random record number written but, unlike the sequential write, the record number is not incremented ready for the next sequential write. The number of records written is determined by the current value of the multi sector count. If an error occurs in writing to a record, the number of records actually written is returned in register H with a value from 0 to 127.

BDOS function 40, WRITE RANDOM WITH ZERO FILL, is a special random-write command for use with files which contain 'holes'. With sequential files, every record up to the end of the file is written and the last record is determined by the RC field in the last extent. But with 'sparse' files, that is files with 'holes', records are missing. CP/M allocates the disk space in units of datablocks (a datablock contains at least 8 records), but the FCB only contains a record count (RC) for the last record in a (Physical) extent, and therefore there is no method for determining if all records in a datablock have actually been written. To reduce the risk of reading a record that has not actually been written, but 'appeared' because one or more other records in that datablock were actually written, this function will set all the remaining records in a datablock to zero valued bytes when the first record is written in that datablock. Therefore, any program which writes sparse files should use this function 40 and not function 34 when writing. In all other respects thus function performs in exactly the same way as for function 34.

BDOS function 99 TRUNCATE FILE is a new BDOS function which allows a file to be truncated. This function is not included with either MP/M or earlier releases of CP/M and, without this function, files could only be increased in size or deleted. (It was actually possible, but very difficult, to truncate a file by using undocumented features, but these techniques do not work under CP/M Plus making this BDOS function essential).

To Truncate a file, the last record number of the file is passed in R0, R1 & R2 in the range 0-3FFFFh. This record must actually exist on the file, or an error will be returned. Consequently it is not possible to use this file to 'empty' a file completely. Unlike the Read and Write Random Functions, the file should NOT be opened when making this call and if a file has been opened it MUST be closed if any record has been written to the file. The FCB must be initialised with the DR, File name and File Type, as well as R0, R1 & R2. TRUNCATE FILE function searches for an FCB for the specified filename and extent derived from R0, R1 & R2 and with the current user number. If the file is password-protected, then a password must be supplied by placing the password in the first 8 bytes of the current DMA buffer, or by the previous use of BDOS FUNCTION 106 to set a default password.

## 7.8.4. Random access using sequential disk functions

In early version of CP/M, the random access commands did not exist, yet it was, then, perfectly possible to perform all of the random access commands through the sequential access commands. In most random function, BDOS takes the random record number R0, R1 & R2 and converts these into the extent number and current record number, and then performs the equivalent sequential access including closing and opening as the extents change. Thus it is still possible to simulate this BDOS conversion in the calling program.

The following code illustrates the calculation:

```

ld hl,r012 ; r012 contains 24 bit number
call bits026 ; extract bits 0-6 returns in <A>
ld (fcb.cr),a
call shift7 ; divide by 128 returns in <DE>
ld a,e
and 00011111b
ld (fcb.ex),a
ex de,hl
add hl,hl
add hl,hl
add hl,hl
ld a,h
and 00111111b
ld d,a
ld hl,fcb.s2
ld a,(hl) ; ABSOLUTELY VITAL
and not 00111111b ; to preserve 2 high bits
or d
ld (hl),a
.....

bits026:
ld a,(hl)
and 01111111b
ret

shift7:
ld a,(hl)
add a,a
inc hl
ld a,(hl)
adc a,a
ld e,a
inc hl
ld a,(hl)
adc a,a
ld d,a
ret

r012: ds 3 ; 24 bit random record number

fcb: ; FCB for sequential access
fcb.dr: ds 0
fcb.fn: ds 11
fcb.ex: db 0
db 0

```

```
fcbl.s2:  db  0
fcbl.rc:  db  0
          ds  16
fcbl.cr:  db  0
```

However the OPEN and MAKE functions set the extent overflow bits in S2 to zero, thus it is not possible to OPEN an extent greater than 31. Under CP/M 2.2, the MAKE function did not check for extent folding, and this could lead to duplicate directory entries when extent folding existed. But, assuming the file was created using sequential access, then Random Reads and Updates can be performed using sequential access.

The earlier CP/M 1.4 produced additional problems, as a write to the 128th record in an extent would automatically lead to the current extent being closed and the next one opened. For this reason, and also because the CP/M BDOS system is not noted for its speed in accessing the disc, some programs take steps to improve the performance. One such method is to open all the extents of a file when the file is first opened and then switch from file to file as the extent boundary is crossed. With the LRU buffering of CP/M Plus this no longer achieves a significant improvement in performance.



## Notes:

1. The directory label is not fully compatible with the MP/M directory label in which bit 4 is used to auto create XFCB when an FCB is created.
2. The bits 4 through 6 of a directory label can only be set if the directory contains SFCBs at every 4th position. If bit 6 is set for access stamp, then setting bit 4 has no effect as the access stamp takes precedence over the create stamp.
3. The directory label does not contain a password if S1 is zero and the password contains either all zeroes or all 020h's. S1 is probably used as the encrypting key.
4. XFCBs can only be created when a directory label exists and when bit 7 of the directory label data byte is set.
5. If the disks are intended for access by other operating systems of the CP/M family, date stamping is not fully compatible across the operating systems MP/M II, DOS Plus, C-DOS and C-CP/M.

## SFCB Format

SFCBs reside in every fourth position of the directory which has been initialised for date and time stamping. In a 128 byte directory record, the first three 32 byte entries are FCBs, XFCBs or directory label, and the fourth 32 byte entry is the SFCB. This SFCB is divided into 3 subfields corresponding to each of the three entries in that directory record as follows:

SFCB FIELDS	
+---+-----+-----+-----+---+	
21  Stamp 1   Stamp 2   Stamp 3	
+---+-----+-----+-----+---+	
Bytes	Description
0	'21H' SFCB identity
01-10	SFCB sub-field for 1st FCB in record
11-20	SFCB sub-field for 2nd FCB in record
21-30	SFCB sub-field for 3rd FCB in record
31	reserved for system use

SFCB sub fields	
Bytes	Description
0 - 3	Create or Access Date and Time Stamp Field
4 - 7	Update Date and Time Stamp Field
8	Password mode
9	Reserved
Notes: The SFCB is only valid if the FCB includes extent 0	
Under MP/M the stamp was part of the XFCB	
The password mode is a copy of the XFCB password mode	

## XFCB Extended FCB

Each file may have an extended XFCB associated with it. The XFCB primarily contains the password although, under MP/M, the XFCB also contains the date stamp. Because the XFCB can reside on a different directory record to the FCB for extent zero, the frequent access of the XFCB and FCB together resulted in a noticeable deterioration of performance. So, under CP/M Plus, the date stamp is transferred to the SFCB and a copy of the password mode is also held on the SFCB. As the SFCB is always in the same directory record as the FCB for extent zero, the performance is improved.

A file name on a disc is not only identified by the name and type of the file, but also the user number; therefore the XFCB must also contain the same name, type and user number of the FCB to which it relates.

XFCBs may be created at the same time as the FCB when the latter is created by setting attribute bit F6' and supplying the password and password mode in the current DMA buffer. If an attempt is made to assign a password to an extent other than physical extent zero, this will result in a BDOS PASSWORD error. However if an attempt is made to assign a password to make an extent that is within an existing physical extent zero, then any password will be updated without any check being made on the current password.

XFCBs may also be created or updated through the BDOS function 103; Write File XFCB. If a password already exists, BDOS will update the password so long as the existing password is correctly supplied.

BDOS function 103 passes a 'logical' XFCB, whilst a search of the directory will return the 'physical' XFCB. The differences are included in the following table:

XFCB Fields		
Bytes	Name	Description
0	dr	Logical XFCB - Drive number Physical XFCB - '1?'H XFCB identity, where ? is user no. (0-15)
01-08	name	Filename as for FCB
09-11	type	Filetype as for FCB
12	pm	Password mode
13-15	s1,s2,rc	Reserved
16-23	password	8 byte password field (encrypted)
24-27	reserved	Reserved
24-27	ts1	MP/M only: 'Create' time and date stamp
28-31	ts2	MP/M only: 'Update' time and date stamp

XFCB - Password Mode	
Bit	Description
7	READ password protected (includes WRITE & DELETE)
6	WRITE password protected (includes DELETE)
5	DELETE password protected

### Directory FCB and Extent Folding

The main entry in the directory is the directory FCB which describes the 'physical' FCB extent. There are a number of differences between the 'physical' FCB and the 'logical' FCB passed to BDOS Functions. The items that differ are:

Byte	Description
0	Logical FCB - Drive number
	Physical FCB - Type of directory entry and user no.
12&14	Logical FCB - Extent number (0-2047)
	Physical FCB - Highest extent number included in FCB
15	Logical FCB - Number of records in FCB
	Physical FCB - Number of records in highest extent

The different use of the 'dr' first byte between the 'logical' and 'physical' FCBs is very distinct as the logical FCB must specify the drive on which the FCB resides, whilst the directory FCB cannot know which drive it is, but needs to identify the user number for the FCB. Consequently this difference leads to little confusion once it is understood. In contrast, the differences between the extent fields and the RC fields between the 'logical' and 'physical' FCB is based on a complex relationship which can be very confusing. The key to the differences is 'EXTENT FOLDING' which is the method by which one or more 'logical' extents are 'folded' into one 'physical' extent. When the DPB (Disk Parameter Block) for a drive has a datablock size of 1k bytes, then there is no extent folding and the Extent and RC fields in the Logical and Physical FCB are identical (except for byte 0). The amount of extent folding can be calculated from the EXM (extent mask) field in the drives DPB. For larger datablocks sizes it is usual for the EXM (extent mask) byte in the DPB to be non-zero, and its value determines the maximum number of 'logical' extents contained in one 'physical' extent. The number of folds is given as EXM+1.

One effect of extent folding is to reduce the number of directory entries as illustrated in the following example:



Directory entries for a sequential file of 80k for drives with different values of EXM					
	EXM=0	EXM=1	EXM=3	EXM=7	EXM=15
Number of entries:	5	3	2	1	1
EX in 1st entry:	0	1	3	4	4
2nd entry:	1	3	4	-	-
3rd entry:	2	4	-	-	-
4th entry:	3	-	-	-	-
5th entry:	4	-	-	-	-

For the example when EXM=1, there are two extent folds therefore each extent describes 32k of data rather than the 16k of data described in the 'logical' extent. As the file is greater than 32k in size, the value of EX in the first extent is set to the maximum value, namely 1. The second entry describes the 2nd 32k of data and as the file is greater than 64k in size, the value of EX in the second extent is set to the maximum value namely 3. But in the 3rd entry which describes the 3rd 32k of data, this exceeds the size of the actual file. The value of EX is then set to the value in the largest 'logical' FCB which in this example would be 4. In the first 2 directory entries, the value of RC is 128, and in the 3rd entry it is also set to 128 as this is the number of records in extent number 4.

Extent folding also impacts the value of the RC (Record Count) field in the FCB. As there is only one value of RC in a 'physical' extent, this value can only be applied to one of the 'logical' extents contained in this 'physical' extent. Thus if for example the 'physical' extent has 8 extent folds, and the value of RC is 5 then if the value of EX is 11, the logical values of RC are as follows:

Unfolding of RC for a file with RC=5, EX=11 and EXM=7		
	Logical extents	
	EX	RC
	----	----
1.	8	128
2.	9	128
3.	10	128
4.	11	5

For the first 3 'logical' extents the value of RC is set to the maximum value of 128 because the actual number is unknown.

#### DIRECTORY SEARCH

Using the above information it is possible to search the directory for all types of directory entries.

The BDOS functions are:

1. BDOS Function 17 - Search first
2. BDOS Function 18 - Search next

The BDOS function 18, Search next, must follow the BDOS function 17 without any intervening BDOS disc functions. The Search first uses the file specification in the passed FCB as a mask with which it uses to search for the first occurrence, and the Search Next uses this same FCB mask to search for the next occurrence. The file specification may include wildcards for the dr, name, type and extent fields. The address of the FCB is passed as a parameter on a Search First, BDOS saves the pointer to the FCB for any subsequent Search Next, and calculates the number of bytes in the FCB to use for the search. After locating a matching directory entry with either the search first or search next, the position in the directory is saved, ready for the next search.

Unlike the BDOS functions OPEN and MAKE, the two SEARCH functions do not always set the FCB field S2 to zero; instead, if the EX field is a wildcard, then the field S2 must be initialised as it is then also used as part of the matching fields.

### Searching for a file

The standard search is to specify an FCB with optional wildcards with the following fields completed:

Search for files - FCB parameters	
Byte	Value
00	Specify drive (1-16) or 0 for default drive
01-08	Filename characters - wildcard ? may be used
09-11	Filetype characters - wildcard ? may be used
10	EX (0-31) - wildcard ? may be used
12	If EX = '?', S2 (0-63) - wildcard ? may be used
	otherwise search only includes files with S2 = 0

This FCB is passed to BDOS when the SEARCH FIRST is called, and must be maintained for all subsequent SEARCH NEXT calls. The BDOS SEARCH FIRST remembers the address of the FCB passed ready for any subsequent SEARCH NEXT calls. BDOS returns with a directory code in register A (0-3) or a value of 0FFh to indicate that no file was found. (Under CP/M 1.4 the directory code was in the range 0-63).

The search masks out the attribute bits of the PHYSICAL FCB and first checks the PHYSICAL FCB against the current user number. The search on the extent field (EX) excludes any extent folding bits as defined by EXM on the DPB (disk parameter block). Thus no conflict arises because the PHYSICAL FCB may include more than one LOGICAL extents.

The match against S2 is only made if EX is set to a '?', otherwise only directory FCBS with S2 = 0 are included in the search irrespective of the supplied value of S2.

On a successful return, BDOS also copies the directory record in which the filename was found, into the current DMA buffer. The directory record comprises 4 entries, and the start address of the directory FCB for the file found can be calculated by multiplying the directory number (0-3) by 32 and adding this to the address of the DMA buffer. The usual method is as follows:

```

add a,a
add a,a
add a,a
add a,a
add a,a
ld l,a
ld h,0      ; offset to FCB
ld de,dbuff ; current DMA buffer
add hl,de   ; HL is start of matching FCB

```

The FCB pointed at is the PHYSICAL FCB which is different from the LOGICAL FCB passed to the BDOS by a calling program. It will also contain any attributes set

#### Directory Scan

A special version of the directory search is to search for all the directory entries. This is necessary to search for directory labels, SFCBs and XFCB as well as erased files. This search requires only the first byte to be specified as a wildcard '?' in the passed FCB.

Scan Directory Search - FCB parameters	
Byte	Value
00	'?'

This FCB is passed to BDOS when the SEARCH FIRST is called, and must be maintained for all subsequent SEARCH NEXT calls. The BDOS SEARCH FIRST remembers this special form of the search ready for any subsequent SEARCH NEXT calls. The search is made on the current default drive, and regardless of the user number. Even though the BDOS remembers this special search, the SEARCH NEXT still examines the first byte of the FCB. Changing this byte before a SEARCH NEXT can have unexpected consequences.

BDOS returns with a directory code in register A (0-3) or a value of 0FFh to indicate the end of the directory found. (Under some versions of CP/M which maintain a 'high water' mark of active FCBs, this end of directory may exclude some remaining erased entries).

The search FIRST returns the first directory entry on the current default drive. The search NEXT returns the next directory entry on the current default drive unless the end of the directory is found.

As before, on a successful return, BDOS also copies the directory record containing the directory entry into the current DMA buffer and this entry can be extracted using the same method.

The calling program must test the first byte of the entry to determine the type of directory entry returned.

## 7.8.6. Special functions

A full directory analysis requires additional information such as the data block size and the extent folding factor. Associated with each logical drive is the Disk Parameter Block (DPB), the Disk Parameter Header (DPH), and the Allocation Vector. Under normal use, these should never require access, and the only BDOS Function of general use is 46, Get Disk Free Space. However, an example of the use for this additional information is the utility SHOW and in the program DISKSTAT (In the UK User Group library) which displays the values of the DPB and the DPH and provides a simple analysis the current allocation. This program allows the user to examine the characteristics of the disc system which can assist in transferring data from one machine to another. A study of the program will illustrate the techniques necessary to examine these items.

Each drive has associated with it a Disk Parameter Header. This cannot be accessed directly through a BDOS Function call, but can only be accessed indirectly by using BDOS Function 50 for making a direct BIOS call to select the required disk which returns the address of the Disk Parameter Header.

Although the Disk Parameter Header includes the address of both the Disk Parameter Block and the Allocation Vector, these are available directly through BDOS functions. BDOS Function 32 GET DPB ADDR and BDOS Function 27 GET ALLOC ADDR returns the address. Under CP/M Plus, the ALLOC ADDR is usually an address in a different bank of memory (Bank 0), and not in the callers memory (TPA) bank (Bank 1): This makes it exceedingly difficult (but not impossible) to access the allocation vector.

The BDOS Function 32 GET DPB ADDR returns the address of the 17 byte (15 under CP/M 2.2) Disk Parameter Block. The fields are fully detailed in the section describing direct BIOS calls, but the most useful items are the following:

Disk Parameter Blocks - Most useful fields		
Byte	Name	Description
02	BSH	Block Shift factor, from which the Blocksize is calculated as $128 * (2^{BSH})$ bytes, or as $2^{BSH-3}$ kilo bytes.
04	EXM	Extent mask to determine extent folding in directory FCB, where $EXM + 1$ is the number of LOGICAL 16k EXTENTS contained in one PHYSICAL directory EXTENT.
05-06	DSM	The number of the largest datablock, where $DSM+1$ determines the total storage capacity measured in blocksize units
07-08	DRM	The number of the largest directory entry, where $DRM+1$ determines the total number of directory entries
09-10	ALO/1	A 16 bit mask to determine which of the first 16 datablocks are reserved for the directory (Bit 7 of byte 09 corresponding to datablock 0 must always be set)

Using these items, it is possible, for example, to build one's own allocation vector for analysing the directory use. The DSM not only determines the size of the datablocks, but it also determines the field size of the FCB datablock assignment. If the HIGH byte of DSM is zero, then the FCB uses 16 x 8 bit assignments, whilst if the HIGH byte of DSM is not zero, the FCB uses 8 x 16 bit assignments.

---

BDOS FUNCTION 15 - Open File FCB

---

## Calling parameters

Register C = 0FH  
 Register DE - FCB address (33 bytes)  
 FCB dr,n,t - Unambiguous file specification  
 FCB ex = 00H  
 FCB cr = 00H ready for sequential file access  
           = 0FFH to request last record count  
 DMA Buffer - Password if FCB is password protected  
 Password - 8 character password (see function 106)

If the file is either read or write password protected,  
 the current password is first tested and if this fails,  
 the password in the current DMA buffer is then tested.

## Returned parameters

Register A - Directory code (0-3)  
 FCB - Completed with found file and 'opened'  
 FCB f8' - Set if file opened under user 0 & user > 0  
 FCB f7' - Set if incorrect write protect password  
 FCB cr - Last record count if requested

## Error returns

Register A = FFH  
 Register H = 0 No file found  
           or H - Physical error in extended error mode

## Physical Error returns (extended disc error mode only)

= 01H: Disk I/O error  
 = 04H: Invalid drive (drive does not exist)  
 = 07H: Read mode password protect password error  
 = 09H: Wildcard ? in the FCB filename or filetype fields

---

+-----+  
| BDOS FUNCTION 16 - Close File FCB |  
+-----+

## | Calling parameters |

| Register C = 10H |

| Register DE - FCB address (opened) |

| FCB f5' - Set if FCB to remain opened |

| DMA Buffer - Password if FCB is password protected |

| Password - 8 character password (see function 106) |

|  
| If the file is password protected, the current password  
| is first tested and if this fails, the password in the  
| current DMA buffer is then tested. |

## | Returned parameters |

| Register A - Directory code (0-3) |

## | Error returns |

| Register A = FFH |

| Register H = 0 No file found |

| or H - Physical error in extended error mode |

## | Physical Error returns (extended disc error mode only) |

| = 01H: Disk I/O error |

| = 02H: Read/Only disk |

| = 04H: Invalid drive (drive does not exist) |  
+-----+

BDOS FUNCTION 17 - Search For First FCB
<p>Calling parameters</p> <p>Register C = 11H</p> <p>Register DE - FCB address</p> <p>FCB dr,n,t - File specification including wildcards</p> <p>FCB ex,s2 - Extent for search including wildcards</p> <p>Returned parameters</p> <p>Register A - Directory code (0-3)</p> <p>Current DMA - 128 byte directory record containing FCB</p> <p>Error returns</p> <p>Register A = FFH</p> <p>Register H = 0 No matching FCB found</p> <p>or H - Physical error in extended error mode</p> <p>Physical Error returns (extended disc error mode only)</p> <p>= 01H: Disk I/O error</p> <p>= 04H: Invalid drive (drive does not exist)</p> <p>Additional functions</p> <p>If the FCB.DR byte is set to the wildcard '?', search first/next returns each directory entry without matching for the current default drive. The FCB.S2 extent field can only be set if the FCB.EX is a wildcard '?'.</p>

BDOS FUNCTION 18 - Search For Next FCB
<p>Calling parameters</p> <p>Register C = 11H</p> <p>DE in FIRST - FCB with matching fields</p> <p>Returned parameters (see BDOS FUNCTION 17 - Search FIRST)</p> <p>Register A - Directory code (0-3)</p> <p>Current DMA - 128 byte directory record containing FCB</p> <p>Error returns (see BDOS FUNCTION 17 - Search FIRST)</p> <p>Register A = FFH</p> <p>Register H = 0 No matching FCB found</p> <p>or H - Physical error in extended error mode</p>



---

BDOS FUNCTION 19 - Delete File FCB

---

## Calling parameters

Register C = 13H  
 Register DE - FCB address  
 FCB dr,n,t - File specification with wildcards  
 FCB f5' - Set to delete XFCB's only  
 DMA Buffer - Password if FCB is password protected  
 Password - 8 character password (see function 106)

## Returned parameters

Register A - Directory code (0-3)

If the file is either read or write password protected, the current password is first tested and if this fails, the password in the current DMA buffer is then tested.

## Error returns

Register A = FFH  
 Register H = 0 No file found  
 or H - Physical error in extended error mode

## Physical Error returns (extended disc error mode only)

= 01H: Disk I/O error  
 = 02H: Read only disk  
 = 03H: Read only file  
 = 04H: Invalid drive (drive does not exist)  
 = 07H: One or more FCB's or XFCB's failed password check

---

---

BDOS FUNCTION 20 - Read Sequential

---

## Calling parameters

Register C = 14H  
 Register DE - FCB address (opened)  
 FCB cr = Next record number to be read in extent  
 DMA Buffer - One or more blocks of 128 byte records  
 SCB - Sector Count set to number of sectors

## Returned parameters - Successful read

Register A = 00H:

## Error returns - Standard errors

Register A = 01H: At EOF, or reading unwritten data  
 = 09H: Invalid FCB, not opened  
 = 10H: Media changed whilst file opened  
 Register H = 00H-07FH: Number of sectors read

## Error returns - Extended or Physical errors

Register A = FFH  
 Register H - Physical error in extended error mode

## Physical Error returns (extended disc error mode only)

= 01H: Disk I/O error  
 = 04H: Invalid drive (drive does not exist)

---

## BDOS FUNCTION 21 - Write Sequential

## Calling parameters

Register C = 15H  
 Register DE - FCB address (opened)  
 FCB cr = Next record number to write in extent  
 DMA Buffer - One or more blocks of 128 byte records  
 SCB - Sector Count set to number of sectors

## Returned parameters - Successful write

Register A = 00H

## Error returns - Standard errors

Register A = 01H: Directory space full  
               = 02H: Datablocks full  
               = 09H: Invalid FCB, not opened  
               = 10H: Media changed whilst file opened  
 Register H = 00H-07FH: Number of sectors written

## Error returns - Extended or Physical errors

Register A = FFH  
 Register H - Physical error in extended error mode

## Physical Error returns (extended disc error mode only)

= 01H: Disk I/O error  
 = 02H: Read Only Disk  
 = 03H: Read Only File or  
       FCB f8' is set - file opened on user 0 & user > 0  
       FCB f7' is set - incorrect write protect password  
 = 04H: Invalid drive (drive does not exist)

## BDOS FUNCTION 22 - Make File FCB

## Calling parameters

Register C = 16H  
 Register DE - FCB address  
 FCB dr,n,t - Unambiguous file specification  
 FCB ex = 00H  
 FCB f6' = Set to assign password to file (XFCB)  
 DMA Buffer - Password & password mode if FCB f6' set  
 Password - 8 character password  
 Pswrd. mode - 1 byte mode (see function 102)

File password protection can only be implemented if the disk contains a directory label enabeling passwords. Otherwise the password attribute FCB f6' is ignored.

## Returned parameters

Register A - Directory code (0-3)  
 FCB - Created as new file and 'opened'  
 XFCB - Initialised if password required

## Error returns

Register A = FFH  
 Register H = 0 No directory space  
 or H - Physical error in extended error mode

## Physical Error returns (extended disc error mode only)

= 01H: Disk I/O error  
 = 02H: Read Only disk  
 = 04H: Invalid drive (drive does not exist)  
 = 08H: File already exists with same user number  
 = 09H: Wildcard ? in the FCB filename or filetype fields

---

BDOS FUNCTION 23 - Rename File

---

## Calling parameters

Register C = 16H  
 Register DE - FCB address  
 FCB 00H = drive (0,1-15) to select  
 FCB 01H-0BH = Old file name and type  
 FCB 11H-1BH = New file name and type  
 DMA Buffer - Password if FCB is password protected  
 Password - 8 character password (see function 106)

## Returned parameters

Register A = 0

If the file is either read or write password protected, the current password is first tested and if this fails, the password in the current DMA buffer is then tested.

## Error returns

Register A = FFH  
 Register H = 0 No file found  
 or H = Physical error in extended error mode

## Physical Error returns (extended disc error mode only)

= 01H: Disk I/O error  
 = 02H: Read Only disk  
 = 03H: Read Only file  
 = 04H: Invalid drive (drive does not exist)  
 = 07H: File password protected and password error  
 = 08H: New File already exists with same user number  
 = 09H: Wildcard ? in the FCB filename or filetype fields

---

BDOS FUNCTION 26 - Set DMA Transfer Address
<p>Calling parameters</p> <p>Register C = 1AH</p> <p>Register DE - DMA address</p> <p>DMA = Data buffer of one or more 128 byte records</p> <p>No Returned parameters</p> <p>The SET DMA changes the current DMA address for all BDOS functions that use the DMA address, except for Function 47 which always uses the CCP default of 0080H.</p> <p>The DMA buffer must be sufficient for the BDOS functions, and for the disk transfer functions, must include space for any multiple sector transfers.</p>

BDOS FUNCTION 27 - Get address ALLOC vector

Calling parameters  
Register C = 1BH

Returned parameters - drive logged in  
Register HL - Address of allocation vector

Error returns - in extended error mode only  
Register HL = 0FFFFH for physical error

The address returned is the allocation vector for the current default drive. If this drive is not logged in, it is first logged in.

As the allocation vector is usually located in the system bank and not the users TPA bank, the address cannot be used to access the allocation vector directly.

This function is of little value under CP/M Plus, See BDOS function 46 Get Disk Free Space.

## BDOS FUNCTION 30 - Set FCB file attributes

## Calling parameters

Register C = 1EH  
 Register DE = FCB address  
 FCB dr,n,t - Unambiguous file specification  
 FCB f1' = F1 attribute (user defined)  
 FCB f2' = F2 attribute (user defined)  
 FCB f3' = F3 attribute (user defined)  
 FCB f4' = F4 attribute (user defined)  
 FCB f6' = 1: Set byte count supplied in FCB.CR  
 FCB CR = Byte count if f6' set  
 FCB t1' = File Read-Only attribute  
 FCB t2' = System attribute (see DIR and DIRS)  
 FCB t3' = Archive attribute (see PIP archive option)  
 DMA Buffer - Password if FCB is password protected  
 Password - 8 character password (see function 106)

If the file has any password protection, the current password in first tested and if this fails, the password in the current DMA buffer is then tested.

## Returned parameters

Register A - Directory code (0-3)  
 FCB - Extent 0 of FCB updated with attribute and S1 update with CR if f6' set

## Error returns

Register A = FFH  
 Register H = 0 No file found  
 or H - Physical error in extended error mode

## Physical Error returns (extended disc error mode only)

= 01H: Disk I/O error  
 = 02H: Read Only disk  
 = 04H: Invalid drive (drive does not exist)  
 = 07H: File password error  
 = 09H: Wildcard ? in the FCB filename or filetype fields



```
+-----+
| BDOS FUNCTION 31 - Get address DPB parameter block |
+-----+
| Calling parameters |
|   Register C = 1FH |
|
| Returned parameters - drive logged in |
|   Register HL - Address of DPB parameter block |
|
| Error returns - in extended error mode only |
|   Register HL = 0FFFFH for physical error |
|
| The address returned is the Disk Parameter Block (DPB) |
| current default drive. If this drive is not logged in |
| it is first logged in. |
+-----+
```

BDOS FUNCTION 33 - Read Random
Calling parameters
Register C = 21H
Register DE - FCB address (opened)
FCB r0,r1,r2- Random Record number to be read
DMA Buffer - One or more blocks of 128 byte records
SCB - Sector Count set to number of sectors
Returned parameters - Successful read
Register A = 00H
FCB EX, CR - 'opened' extent
Error returns - Standard errors
Register A = 01H: At EOF, or reading unwritten data
= 03H: Cannot close current extent (DIR FULL)
= 04H: Cannot open required extent (missing)
= 06H: Random record number > 3FFFFH
= 10H: Media changed whilst file opened
Register H = 00H-07FH: Number of sectors read
Error returns - Extended or Physical errors
Register A = FFH
Register H - Physical error in extended error mode
Physical Error returns (extended disc error mode only)
= 01H: Disk I/O error
= 04H: Invalid drive (drive does not exist)

---

 BDOS FUNCTION 34 - Write Random
 

---

## Calling parameters

Register C = 22H  
 Register DE - FCB address (opened)  
 FCB r0,r1,r2- Random Record number to be written  
 DMA Buffer - One or more blocks of 128 byte records  
 SCB - Sector Count set to number of sectors

## Returned parameters - Successful write

Register A = 00H  
 FCB EX, CR - 'opened' extent

## Error returns - Standard errors

Register A = 02H: Datablocks full (disk full)  
               = 03H: Cannot close current extent (dir full)  
               = 05H: Cannot open required extent (dir full)  
               = 06H: Random record number > 3FFFFH  
               = 09H: Invalid FCB, not opened  
               = 10H: Media changed whilst file opened  
 Register H = 00H-07FH: Number of sectors read

## Error returns - Extended or Physical errors

Register A = FFH  
 Register H - Physical error in extended error mode

## Physical Error returns (extended disc error mode only)

= 01H: Disk I/O error  
 = 02H: Read Only Disk  
 = 03H: Read Only File or  
           FCB f8' is set - file opened on user 0 & user > 0  
           FCB f7' is set - incorrect write protect password  
 = 04H: Invalid drive (drive does not exist)

---

---

BDOS FUNCTION 35 - Compute File Size

---

## Calling parameters

Register C = 23H

Register DE - FCB address (may be opened, 36 bytes)

FCB dr,n,t - Unambiguous file specification

## Returned parameters

Register A - 00H

FCB r0,r1,r2- Random Record Number of the next record  
after the EOF.

## Error returns

Register A = FFH

Register H = 0 No file found

or H - Physical error in extended error mode

## Physical Error returns (extended disc error mode only)

= 01H: Disk I/O error

= 04H: Invalid drive (drive does not exist)

= 09H: Wildcard ? in the FCB filename or filetype fields

The returned random record number is the file size is 128  
byte units. The storage space occupied by this file may  
be different than this file size, because the last record  
may not fall on a datablock boundary, and because the file  
may not contain missing datablocks if it was created  
randomly

---

BDOS FUNCTION 36 - Set Random Record
Calling parameters
Register C = 24H
Register DE - FCB address (need not be opened, 36 byte)
FCB ex & s2 = Initialised to extent number
FCB cr = Initialised to next record in extent
Returned parameters
Register A - 00H
FCB r0,r1,r2- Random Record Number of the next record
Typically the FCB passed is an opened file which has already been accessed sequentially. This function returns the random record number of the next sequential record.

+-----+  
| BDOS FUNCTION 40 - Write Random with Zero Fill |  
+-----+

| Calling parameters |

|     Register C = 28H |

|     Register DE - FCB address (opened) |

| The passed FCB and the returned parameters are identical |  
| to the BDOS Function 34 Write Random. |

| This function is identical to the Write Random, but with |  
| the added feature of filling any new datablock with zeroes |  
| before the record is written. A new datablock is allocated |  
| when the random record references an unallocated datablock. |  
| Using Function 34, any new datablocks allocated contain |  
| uninitialized data. |

+-----+	
	BDOS FUNCTION 44 - Set Multi-Sector count
+-----+	
	Calling parameters
	Register C = 2CH
	Register E = Number of sectors (1 - 128)
	Returned parameters
	Register A = 00H: Number of sectors in range 1 - 128
	= FFH: Error in number of sectors
	SCB           = Updated with multi-sector count
	The number of sectors is initialised to 1 by the CCP.
+-----+	

BDOS FUNCTION 45 - Set BDOS disk error mode	
Calling parameters	
Register C	= 2DH
Register E	= BDOS error mode
Returned parameters	
SCB	= Updated with BDOS error mode
The BDOS error mode controls the message and processing of disc functions that cause either an extended error, or a physical error.	
Error Mode	Error processing
-----	-----
00H	Default mode, error causes warm boot
00H-FDH	BDOS message displayed, then warm boot
FEH	BDOS message, returns extended error no.
FFH	No BDOS message, returns extended error no.



BDOS FUNCTION 46 - Get Disk Free Space

Calling parameters

Register C = 2EH

Register E - Drive number

Drive no. = 0,1 thru 15 for drive A:, B:, thru P:

Returned parameters

Register A = 0

DMA Buffer - Free space

Free Space is returned in the current default buffer in the same format as R0, R1 & R2 in the FCB parameters.

Byte Offset 0 - Low byte of free space

Byte Offset 1 - Middle byte of free space

Byte Offset 2 - High byte of free space

Note: The free space calculation may be in error if the drive is marked read-only. This often occurred under CP/M 2.2, but is unlikely under CP/M Plus.

Error returns - in extended error mode only

Register A = FFH

Register H - Physical error in extended error mode

Physical Error returns (extended disc error mode only)

= 01H: Disk I/O error

= 04H: Invalid drive (drive does not exist)

+-----+  
| BDOS FUNCTION 48 - Flush Buffers |  
+-----+

## | Calling parameters |

| Register C = 30H |

| Register E - Purge flag |

| Purge Flag = 0FFH to purge any data buffers after write  
| to disk. |

| &lt; 0FFH the data buffers retain data. |

## | Returned parameters |

| Register A = 0 |

## | Error returns - in extended error mode only |

| Register A = FFH |

| Register H - Physical error in extended error mode |

## | Physical Error returns (extended disc error mode only) |

| = 01H: Disk I/O error |

| = 02H: Read Only disk |

| = 04H: Invalid drive (drive does not exist) |

| CP/M Plus maintains LRU buffering of data transfer to the  
| disk system. These buffers are provided for any blocking  
| and deblocking of data and for any LRU buffering of data  
| to reduce the disk activity overhead to the system.  
| The FLUSH command ensures that any data buffers containing  
| data waiting for write to disk are written. The Purge  
| ensures that the next read returns data from the disk. |  
+-----+

BDOS FUNCTION 98 - Free Temporary Datablocks

Calling parameters

Register C = 62H

Returned parameters

Register A = 00H

Error returns - extended error mode only

Register A = FFH

Register H - Physical error in extended error mode

Physical Error returns (extended disc error mode only)

= 04H: Invalid drive (drive does not exist)

As a new datablock is allocated to a file during disc write functions, the allocation is marked in a temporary table. Only when the FCB extent is closed, implicitly by writing to another extent, or explicitly by a close, the temporary allocation is copied to the permanent allocation table. This function copies the permanent allocation table over the temporary allocation table, thereby removing any temporary allocation. CCP makes this call when programs terminate. This function should only be called after all opened permanent files have been closed.

Under CP/M 2.2, and with CP/M Plus systems which have not configured to support a temporary datablock allocation table, any temporary file space is only returned following a drive reset, or CTRL-C disk reset.

## BDOS FUNCTION 99 - Truncate File FCB

## Calling parameters

Register C = 83H  
 Register DE - FCB address (un-opened)  
 FCB dr,n,t - Unambiguous file specification  
 FCB r0,r1,r2- Random Record number of last record  
 DMA Buffer - Password if FCB is password protected  
 Password - 8 character password (see function 106)

If the file has any password protection, the current password in first tested and if this fails, the password in the current DMA buffer is then tested.

## Returned parameters

Register A - Directory code (0-3)

## Error returns

Register A = FFH  
 Register H = 0 No file found, or record not found  
 or H - Physical error in extended error mode

## Physical Error returns (extended disc error mode only)

= 01H: Disk I/O error  
 = 02H: Read Only disk  
 = 03H: Read Only File  
 = 04H: Invalid drive (drive does not exist)  
 = 07H: File password error  
 = 09H: Wildcard ? in the FCB filename or filetype fields

The Record specified by the random record number must exist for the file to be truncated. This record then becomes the last record on the file. This function also cancels the 'opened' status of any FCB passed, which will prohibit further file access until the FCB is 'opened' again. It is best to close the file before the function is called.

## BDOS FUNCTION 100 - Set Directory Label

## Calling parameters

Register C = 64H  
 Register DE - FCB address  
 FCB dr,n,t - Initialised to drive and label name & type  
 FCB ex - Directory Label Data Byte

## Directory Label Data Byte

Bit 7 - Enable password protection of files  
 Bit 6 - Enable access date and time stamp of files  
 Bit 5 - Enable update date and time stamp of files  
 Bit 4 - Enable create date and time stamp of files  
 Bit 0 - Assign a new password to directory label

DMA Buffer - Password1 if FCB is password protected  
 Offset 08 - Password2 if Data Byte bit 0 set  
 Password1 - 8 character password (see function 106)  
 Password2 - new 8 character password

If the current directory label is password protected, the current password is first tested and if this fails, the password in the current DMA buffer is then tested.

## Returned parameters

Register A - Directory code (0-3)

## Error returns

Register A = FFH  
 Register H = 0 No directory space or no SFCB  
 or H - Physical error in extended error mode

## Physical Error returns (extended disc error mode only)

= 01H: Disk I/O error  
 = 02H: Read Only disk  
 = 04H: Invalid drive (drive does not exist)  
 = 07H: File password error

Only one directory label can exist in a drive's directory, and the directory label password, if assigned, cannot be circumvented, whereas file password protection can be disabled with the directory label data byte. If the new directory password is all blanks, the directory label password protection is removed.

---

 BDOS FUNCTION 101 - Return Directory Label Data Byte
 

---

## Calling parameters

Register C = 65H

Register E - Drive (0-15, where 0 = A: and 15 = P:)

## Returned parameters

Register A - Directory Label Data Byte (01H-F1H)

## Directory Label Data Byte

Bit 7 - Enable password protection of files

Bit 6 - Enable access date and time stamp of files

Bit 5 - Enable update date and time stamp of files

Bit 4 - Enable create date and time stamp of files

Bit 0 = 1 if Directory label exists

## Error returns - normal errors

Register A = 00H: No directory Label

## Error returns - Extended errors

Register A = FFH

Register H - Physical error in extended error mode

## Physical Error returns (extended disc error mode only)

= 01H: Disk I/O error

= 04H: Invalid drive (drive does not exist)

---

 BDOS FUNCTION 102 - Read File Date Stamp and Password Mode
 

---

## Calling parameters

Register C = 66H  
 Register DE - FCB address  
 FCB dr,n,t - Unambiguous file specification

## Returned parameters

Register A - Directory code (0-3)  
 FCB 12 - Password Mode Field (=0 if none)  
 FCB 24-27 - Create or Access time stamp (=0 if none)  
 FCB 28-31 - Update time stamp (=0 if none)

## Password Mode Field

Bit 7 - Read, Write & Delete password protected  
 Bit 6 - Write & Delete password protected  
 Bit 5 - Delete password protected

## Time stamp

Byte 0-1 - Date field (16 bit binary)  
 Byte 2 - Hour field (Packed BCD)  
 Byte 3 - Minute field (Packed BCD)

## Error returns

Register A = FFH  
 Register H = 0 No file found  
 or H - Physical error in extended error mode

## Physical Error returns (extended disc error mode only)

= 01H: Disk I/O error  
 = 04H: Invalid drive (drive does not exist)  
 = 09H: Wildcard ? in the FCB filename or filetype fields

---

---

BDOS FUNCTION 103 - Write File Password XFCB

---

## Calling parameters

Register C = 67H  
 Register DE - FCB address  
 FCB dr,n,t - Unambiguous file specification  
 FCB ex - XFCB password mode byte

## Password Mode Byte

Bit 7 - Read mode  
 Bit 6 - Write mode  
 Bit 5 - Delete mode  
 Bit 0 - Assign a new password to directory label

DMA Buffer - Password1 if FCB is password protected

Offset 08 - Password2 if Data Byte bit 0 set

Password1 - 8 character password (see function 106)

Password2 - new 8 character password

If the specified file is currently password protected, the current password is first tested and if this fails, the password in the current DMA buffer is then tested.

## Returned parameters

Register A - Directory code (0-3)  
 XFCB - Created or Updated

## Returned parameters

Register A - Directory code (0-3)  
 XFCB - Created or Updated

## Error returns

Register A = FFH  
 Register H = 0: No directory label exists  
                   or No file found  
                   or No directory space  
                   or Passwords disabled in directory label  
 or H - Physical error in extended error mode

## Physical Error returns (extended disc error mode only)

= 01H: Disk I/O error  
 = 02H: Read Only disk  
 = 04H: Invalid drive (drive does not exist)  
 = 07H: File password error

---



```
+-----+
| BDOS FUNCTION 106 - Set Default Password |
+-----+
| Calling parameters                       |
|   Register C = 6AH                     |
|   Register DE = Password address        |
|   Password   = 8 Byte password         |
|                                         |
| No Returned parameters                  |
|                                         |
| When BDOS accesses an unopened password |
| protected FCB, it checks both this default |
| password and the first 8 bytes in the |
| current DMA buffer. The password is |
| accepted if either matches the file |
| password.                             |
+-----+
```

---

 XDOS FUNCTION 152 - Parse Filename
 

---

## Calling parameters

Register C = 98H

Register DE - FFCB address

## Parse Filename Control Block

Byte 0-1 - Ascii String Address

Byte 2-3 - FCB address (36 bytes)

## Ascii Sting

String of ascii characters containing a file specification terminated with a delimiter

## File Specification

(d:){filename}{.type}{;password}

Each field (enclosed in pair of curly brackets) is optional, but at least one must be specified and where

d: - drive name (A:, B: etc. to P:)

filename - up to 8 character ascii filename

.type - full stop followed by up to 3 characters

;password - semicolon followed by up to 8 characters

## Delimiters

' ' - Space (except any leading spaces and tabs)

09H - Tab (except any leading spaces and tabs)

0DH - Return

00H - Null

';' - Semicolon (except before password field)

'=' - Equal

'&lt;' - Less than

'&gt;' - Greater than

',' - Full Stop (except as filename and type separator)

':' - Except after drive field

',' - Comma

'|' - Vertical Bar

'[' - Left square bracket

']' - Right square bracket

'/' - Slant (MP/M)

'\$' - Dollar (MP/M)

## Returned parameters

Register HL = 0000H: Terminator is NULL or RETURN

&gt; 0000H: Address of terminator

FCB - 36 bytes initialised

## Error returns

Register HL = 0FFFFH: Error in file specification

XDOS FUNCTION 152 - FCB initialization	
FCB Offset	Contents
Byte 00	The drive field set to 1 for A: etc, If no drive specified set to 0
Byte 01-08	Filename converted to upper case, padded to fill 8 characters with blanks, an '*' is replaced with a '?' and all remaining filename characters are filled with '?'
Byte 09-11	Filetype converted to upper case, padded to fill 3 characters with blanks, an '*' is replaced with a '?' and all remaining filename characters are filled with '?'
Byte 12-15	Initialised to zeroes
Byte 16-23	Password converted to upper case, padded to fill 8 characters with blanks
Byte 24-25	(MP/M only) Offset of password in the source string is placed here. Set to zero for none.
Byte 26	Length of password supplied, or zero if none

FCB FIELD INITIALISATION		
Function	FCB fields	Wild card '?'
15 OPEN	DR, F1-F8, T1-T3, EX, CR	
16 CLOSE	'opened' FCB, F5'	
17 FIRST	DR, F1-F8, T1-T3, EX, S2	all
19 DELETE	DR, F1-F8, T1-T3, F5'	F1-F8, T1-T3
20 READ	'opened' FCB, CR	
21 WRITE	'opened' FCB, CR	
22 MAKE	DR, F1-F8, T1-T3, EX, F6'	
23 RENAME	DR, F1-F8, T1-T3, Bytes 17-27	
30 ATTRIBUTES	DR, F1-F8, T1-T3, F1'-F4', T1'-T3', F6'/CR	
33 RD RANDOM	'opened' FCB, R0, R1, R2	
34 WR RANDOM	'opened' FCB, R0, R1, R2	
35 FILE SIZE	DR, F1-F8, T1-T3	
36 SET RANDOM	'opened' FCB, CR	
40 ZERO FILL	'opened' FCB, R0, R1, R2	
41 TST/WRITE	'opened' FCB, R0, R1, R2	
42 LOCK	'opened' FCB, R0, R1, R2	
43 UNLOCK	'opened' FCB, R0, R1, R2	
59 OVERLAY	'opened' FCB, R0, R1	
99 TRUNCATE	DR, F1-F8, T1-T3, R0, R1, R2	
100 SET LABEL	DR, F1-F8, T1-T3, EX	
102 RD DATE	DR, F1-F8, T1-T3	
103 WR XFCB	DR, F1-F8, T1-T3	

SUCCESSFUL RETURN CODES FOR BDOS CALLS THAT USE THE FCB	
Function	Register A returned
15 OPEN	0-3 (directory code)
16 CLOSE	0-3 (directory code)
17 FIRST	0-3 (directory code)
18 NEXT	0-3 (directory code)
19 DELETE	0-3 (directory code)
20 READ	0
21 WRITE	0
22 MAKE	0-3 (directory code)
23 RENAME	0-3 (directory code)
30 ATTRIBUTES	0-3 (directory code)
33 RD RANDOM	0
34 WR RANDOM	0
35 FILE SIZE	0
36 SET RANDOM	0 (always successful)
40 ZERO FILL	0
41 TST/WRITE	0
42 LOCK	0
43 UNLOCK	0
59 OVERLAY	0
99 TRUNCATE	0-3 (directory code)
100 SET LABEL	0-3 (directory code)
102 RD XFCB	0-3 (directory code)
103 WR XFCB	0-3 (directory code)
Note: Under CP/M Plus the directory code is usually 0	

STANDARD ERROR RETURN CODES FOR BDOS CALLS THAT USE THE FCB	
Function	Register A returned (H = 0)
15 OPEN	0FFh - No FCB
16 CLOSE	0FFh - No FCB
17 FIRST	0FFh - No FCB
18 NEXT	0FFh - No FCB
19 DELETE	0FFh - No FCB
22 MAKE	0FFh - Directory full
23 RENAME	0FFh - No FCB (1st)
30 ATTRIBUTES	0FFh - No FCB
35 FILE SIZE	0FFh - No FCB
36 SET RANDOM	no errors
42 LOCK	not implemented
43 UNLOCK	not implemented
59 OVERLAY	0FFh - Loader RSX not resident
	0FEh - No memory to load overlay
99 TRUNCATE	0FFh - No FCB, or bad R0, R1, R2
100 SET LABEL	0FFh - No space, or no XFCB for date stamp
102 RD XFCB	0FFh - No XFCB
103 WR XFCB	0FFh - No XFCB, no FCB, no space, no password

STANDARD MULTI SECTOR ERROR BDOS RETURN CODES	
Function	Register A returned (H = sectors read)
20 READ	01,09,or 10
21 WRITE	01,02,09,or 10
33 RD RANDOM	01,03,04,06,or 10
34 WR RANDOM	02,03,05,06,or 10
40 ZERO FILL	02,03,05,06,or 10
41 TST/WRITE	not implemented

STANDARD BDOS DISC ERROR RETURN NUMBERS IN REGISTER A				
No.	Error	BDOS FUNCTIONS		
1	Reading unwritten data or EOF	20	33	41 42
	No directory space	21		
2	No available data block	21	34	
3	Cannot close current extent		33 34	41 42
4	Seek to unwritten extent		33	41 42
5	Directory full		34	
6	Random record number out of range		33 34	41 42
7	Records did not match			41
8	Record locked by another process			41 42
9	Invalid FCB	20 21		
10	Media change occurred	20 21	33 34	
	or FCB checksum error			41 42
11	Unlocked file verification error			41 42
12	Process record lock limit exceeded			42
13	Bad file ID			42
14	System lock list full			42

EXTENDED/PHYSICAL BDOS DISC ERROR RETURN CODES IN REGISTER H	
NOTES: Extended and physical errors are only returned if the BDOS error mode has been set to 0FFh or 0FEh	
Function	Register H returned (A = 0FFh)
14 SELECT	01, 04
15 OPEN	01, 04, 07, 09
16 CLOSE	01, 02, 04
17 FIRST	01, 04
18 NEXT	01, 04
19 DELETE	01, 02, 03, 04, 07
20 READ	01, 04
21 WRITE	01, 02, 03, 04
22 MAKE	01, 02, 04, 07, 08, 09
23 RENAME	01, 02, 03, 04, 07, 08, 09
30 ATTRIBUTES	01, 02, 04, 07, 09
33 RD RANDOM	01, 04
34 WR RANDOM	01, 02, 03, 04
35 FILE SIZE	01, 04
36 SET RANDOM	no errors
40 ZERO FILL	01, 02, 03, 04
41 TST/WRITE	not implemented
42 LOCK	not implemented
43 UNLOCK	not implemented
59 OVERLAY	01, 04
99 TRUNCATE	01, 02, 03, 04, 07, 09
100 SET LABEL	01, 02, 04, 07
102 RD XFCB	01, 04, 09
103 WR XFCB	01, 02, 04, 07, 09

EXTENDED/PHYSICAL BDOS ERROR RETURN CODES IN REGISTER H	
No. Error (A=0FFh)	BDOS FUNCTIONS
1 - Disk I/O error	14 15 16 17 18 19 20 21 22 23 30 33 34 35 40 59 99 100 102 103
2 - Read/Only disk	16 19 21 22 23 30 34 40 99 100 103
3 - Read/Only file	19 21 23 34 40 99
or f8' set	21
or f7' set	34 40 21
4 - Invalid drive error	14 15 16 17 18 19 20 21 22 23 30 33 34 35 40 59 99 100 102 103
7 - File password error	15 19 22 23 30 100 103
8 - File already exists	22 23
9 - FCB contains wildcard character '?'	15 22 23 30 99 102 103



## 7.9. BDOS clock functions

BDOS provides a clock containing the date and the time of day in the form of a 4 byte TOD structure held in the System Control Block.

+-----+   TOD structure - Time and Date (1st Jan 1978 is day 1)   +-----+	
Bytes 00-01:	Date (16 bit integer)
02:	Hour (8 bit packed BCD digits)
03:	Mins (8 bit packed BCD digits)
04:	Secs (8 bit packed BCD digits)
+-----+	

It is legitimate to access this structure through direct read and write of the System Control Block, rather than through the BDOS Functions 104 and 105 SET and GET TOD. But the TOD structure is only valid following a call to the BIOS to update the clock, and a call to the BIOS to set the clock must follow any changes to the TOD structure. This is because the BDOS clock is a slave to the BIOS clock (if it exists). The BDOS Functions SET and GET do this automatically. The CP/M Plus Utility DATE directly accesses the System Control Block and it fails to make the BIOS call prior to reading the TOD structure during DATE C(ontinuous). Consequently on computer systems which depend on the BIOS updating the clock, the date does not advance.

One advantage of accessing the System Control Block TOD field is because the TOD structure used in the BDOS functions SET and GET does not include the second digit. The SET function sets the date and time with the parameters passed but the seconds are set to zero. The GET function returns the date and time into the parameters block but the seconds are returned in register A. Note that, under MP/M, this function did not return the seconds in register A; instead, MP/M provided an XDOS function 155 as an alternative which included the seconds in the returned TOD structure.

The date is held as a 16 bit integer with day 1 corresponding to January 1st, 1978 which, for those without a 1978 diary, was a Sunday. The day, month and year conversion to date requires the algorithm to adjust for leap years as described in chapter 6.

The time (hours minutes and seconds) are held as 8 bit packed BCD (binary coded decimal) numbers. Arithmetic on a BCD number requires the use of the DAA (Decimal Adjust Instruction) provided as part of the processors instruction set.

A BCD number in register A can be converted to an integer number as follows:

```
ld  b,a
and 0f0h ; mask most significant digit
rrca
rrca
rrca
ld  c,a
add a,a
add a,a
add a,c
ld  c,a
ld  a,b
and 0fh ; mask least significant digit
add a,c ; register A now contains integer number
ret
```

In reverse, an integer number in register A (assumed < 100) can be converted into A BCD number as follows:

```
cp  10
ret c ; if number is less than 10, then same
ld  b,a
xor a
int2bod: inc a
daa
djnz int2bod
ret ; register A now contains packed BCD number
```

The actual mechanism for maintaining the BDOS clock will depend on the hardware configuration of the actual computer. If the computer does not contain a real time clock then, usually, the BDOS clock is advanced every second using hardware interrupts. These computers require the clock to be set when the CP/M Plus system is first loaded.

The more expensive computers contain a real time clock, and the BDOS clock can be maintained with this BIOS clock in two ways:

1. The BDOS clock is set to the BIOS clock when the CP/M Plus system is first loaded, and then the BDOS clock is advanced every second using hardware interrupts.
2. The BDOS clock is set to the BIOS clock whenever the BIOS function call is made to update the BDOS clock.

The BIOS clock can be updated through the BIOS function call to set the BIOS clock to the BDOS clock.

BDOS FUNCTION 104 - SET date and time

Calling parameters

Register C = 68H

Register DE - TOD address

TOD parameter block

Bytes 00-01 = 16 bit integer date

02 = 8 bit packed BCD hour

03 = 8 bit packed BCD minutes

No Returned parameters

The BDOS TOD parameters in the system control block are set to the passed date, hour and minutes. The seconds are initialised to zero. The date is a 16 bit integer with day 1 corresponding to Sunday 1st January 1978.

The hour and minutes are supplied as packed BCD numbers.

BDOS FUNCTION 105 - GET date and time

Calling parameters

Register C = 69H

Register DE - TOD address

TOD parameter block

Bytes 00-01 = 16 bit integer date

02 = 8 bit packed BCD hour

03 = 8 bit packed BCD minutes

Returned parameters

Register A = 8 bit packed BCD seconds (not MP/M)

TOD filled with date, hour and minutes

The BDOS TOD parameters date, hour and minutes in the system control block are copied to the passed TOD parameter block. The seconds are returned in register A.

The date is a 16 bit integer with day 1 corresponding to Sunday 1st January 1978. The hour and minutes are supplied as packed BCD numbers.

## 7.10. BDOS System Control Block

The CP/M Plus BDOS maintains a System Control Block containing 100 bytes of both static and dynamic data. These data items are used for:

1. Communications between the CCP and the BDOS
2. Communications between the BDOS and the BIOS
3. Communications between BDOS and RSX and TPA program
4. Flags, data and pointers used by BDOS and the CCP

The system control block is accessed through BDOS function 49 GET/SET SYSTEM CONTROL BLOCK. The following table summarises the system control block. 'RO' implies that the data can be interrogated by a TPA program, 'RW' implies that the data can be updated by a TPA program, 'BIOS' implies that the data is used in the communication between the BDOS and BIOS; all other items are reserved.

In addition to the SCB, the CCP accesses an area of 12 bytes below the SCB and this is referred to as the XSCB.

SYSTEM CONTROL BLOCK				
Offset	Size	rw/ro	Description	Function
00H	04H			
05H	byte	ro	BDOS version number (31H)	12
06H	04H	rw	Unused for users use	
0AH	06H			
10H	02H	rw	Program error return code	108
12H	byte		Page address of multiple line RSX	
13H	byte		Default Disk	
14H	byte		Default User	
15H	word		Pointer to multiple line in RSX	
17H	8 bits		BDOS-CCP flags	47
18H	8 bits		BDOS-CCP flags includes SETDEF ORDER	
19H	8 bits		BDOS-CCP flags including PROFILE.SUB	
1AH	byte	rw	Console last column number	
1BH	byte	ro	Console column position	
1CH	byte	rw	Console page length	
1DH	01H			
1EH	word		Pointer to command line text	
20H	word		Pointer to command line text	
22H	1 6 bits	rw	CONIN redirection flag	
24H	1 6 bits	rw	CONOUT redirection flag	
26H	1 6 bits	rw	AUXIN redirection flag	
28H	1 6 bits	rw	AUXOUT redirection flag	
2AH	1 6 bits	rw	LSTOUT redirection flag	
2CH	byte	rw	Page mode (=00h)	
2DH	byte			
2EH	byte	rw	CTRL-H active (=00H)	
2FH	byte	rw	RUBOUT active (=00H)	

## 7 - BDOS &amp; BIOS

SYSTEM CONTROL BLOCK (continued)				
Offset	Size	rw/ro	Description	Function
30H	03H			
33H	1 6 bits	rw	Console mode	109
35H	word		Address of 128 byte buffer	
37H	byte	rw	Output delimiter	110
38H	byte	rw	CTRL-P echo to list output flag (=01H)	
39H	03H			
3CH	word	ro	Current DMA address	26,13
3EH	byte	ro	Current Disk	14,25
3FH	word	bios	Pointer to FCB in use by BDOS	
41H	byte	bios	Flag set to 0FFH if FCB above defined	
42H	byte	bios	Bdos function number at last disk error	
43H	01h			
44H	byte	ro	Current User number	32
45H	05H			
4AH	byte	rw	Current Multi sector count	44
4BH	byte	rw	Disk error mode (00H, FEH or FFH)	45
4CH	04H	rw	Drive search chain	
50H	byte	rw	Temporary file drive (0 or 1-16)	
51H	byte	ro	Disk number at last disk error	
52H	02H			
54H	01H	bios	Media flag	
55H	02H			
57H	byte	ro	Flags: (Bit 7 - message, 6 - alloc)	
58H	word	rw	Date stamp - days since 1st Jan '78	104
5AH	byte	rw	Date stamp - hours (BCD)	&
5BH	byte	rw	Date stamp - minutes (BCD)	105
5CH	byte	rw	Date stamp - seconds (BCD)	ditto
5DH	word	ro	Common Memory Base Address	
5FH	03H	bios	Error message jump	
62H	word	ro	Address of current top of User TPA	

EXTENDED SYSTEM CONTROL BLOCK (below SCB)				
Offset	Size	rw/ro	Description	
-02H	02H			
-04H	word	ro	Address of BDOS entry	
-05H	04H			
-09H	04H		Set to zero by CCP before overlay load	

The SCB is loaded down from a page boundary, such that the SCB starts at 100 bytes below the top of a page, or at offset 9CH within a page of memory.

The useful system control block items are:

06H (6) Free, for the use of TPA programs.

Four (4) bytes of data area for use by application programs to save data or flags.

10H (16) Program error return code

The program error code is initialised by CCP to 0000H, and the CCP and SUBMIT uses this code for conditional command line processing. See FUNCTION 108 for details.

12H (18) Page address of multiple line RSX

If a CCP command contains the multiple command delimiter '!', the CCP writes the remainder of the command line to a multiple line RSX. This byte contains the page of this RSX, and the word at 15H contains a pointer to the next character in the multiple line.

13H (19) Default Disk

The default disk (0-15) is the one that is used in the CCP prompt. The CCP sets the current disk to this default disk before loading any program. CCP copies the contents to the low nibble of byte 4 in Page Zero.

14H (20) Default User

The default user (0-15) is the one that is used in the CCP prompt when the user number is not zero. The CCP sets the current user to this default user before loading any program. CCP copies the contents to the high nibble of byte 4 in Page Zero.

15H (21) Pointer to multiple line in RSX

Address of next character in multiple line RSX or zero if no RSX present. Any additional multiple command delimiters '!' are replaced by 0DH, and the line is terminated with a null byte.

17H (23) BDOS-CCP flags

Bit 0 - Set by BDOS when the file \$\$\$SUB exists  
 Bit 1 - Set if COM program contains a NULL COM program  
 Bit 7 - Set following BDOS FUNCTION 47

18H (24) BDOS-CCP flags

Bits 3-4 define type search order if no type supplied:  
 00 - COM  
 01 - COM SUB order  
 10 - SUB COM order  
 11 - PRL COM order  
 Bits 5 & 7 used to control input buffer for CCP COMMAND

19H (25) BDOS-CCP flags

Bit 1 is set after first load of CCP, when 0 CCP searches for file PROFILE.SUB

1AH (26) Console last column number

Byte determines size of screen. For 80 column screen set to 4FH (79).

1BH (27) Console column position

Set by BDOS to current column position on screen. Can be used for screen editing although not maintained by direct console output and does not understand screen emulation commands.

- 1CH (28) Console page length  
Byte determines number of lines to screen. For 24 line screen set to 17H (23). This is used by utilities to display one page at a time if the Page mode (02CH) flag is set to zero.
- 1EH (30) Pointer to command line text
- 20H (32) Pointer to command line text  
BDOS maintains two console character input buffers. One is used whenever the CCP is expecting input, and the other is selected before the CCP loads a program. These buffers contain the last command and can be used in the CTRL-W command. In addition, the BDOS can use a character buffer specified by the CCP for the next command line. The pointers at 1EH and 20H together with the flag bits in the bytes 17 thru 19 in the SCB are used to control the selection of the command line buffers.
- 22H (34) CONIN redirection flag  
The 16 bit redirection flag supports selects one or more of up to 12 physical devices for use as CONIN. Bit 15 corresponds to device 0, and bit 4 corresponds to device 11. The devices are specified in a device table which is returned by bios function 20.
- 24H (36) CONOUT redirection flag  
The 16 bit redirection flag for use as CONOUT to select one or more of up to 12 character devices.
- 26H (38) AUXIN redirection flag  
The 16 bit redirection flag for use as CONOUT to select one or more of up to 12 character devices.
- 28H (40) AUXOUT redirection flag  
The 16 bit redirection flag for use as AUXOUT to select one or more of up to 12 character devices. Often this is set the same as the AUXIN device redirection.
- 2AH (42) LSTOUT redirection flag  
The 16 bit redirection flag for use as LSTOUT to select one or more of up to 12 character devices.
- 2CH (44) Page mode  
A value of 00H in this byte flags page mode, any other value flags continuous display mode. Built In functions and some utilities use this to control the console display.
- 2EH (46) CTRL-H active  
When this byte flag is set to 0FFH then any CTRL-H (backspace) entered from conin will be replaced with a DEL character. This is necessary for non-screen terminals such as a teletypewriter.

2FH (47) RUBOUT active

When this byte flag is set to 0FFH then any RUBOUT/DEL character entered from CONIN will be replaced with a CTRL-H (backspace) character. This is desirable for screen terminals as CP/M console processing interprets the DEL character by indicating its function of deleting the previous character by echoing that character. Whilst this is useful on non-eraseable CONOUT devices it is unrewarding on screens. This byte should normally be set to 0FFH. 33H (51) Console mode 16 bit flags to control the console mode. Normally CCP sets all bits to zero. See BDOS function 109 for details.

35H (53) Address of 128 byte buffer

BDOS maintains a resident buffer for the transfer of data between the resident and banked BDOS. This is not used for disc read and write functions, so the BIOS can use this 128 byte buffer during warm boot reads and writes in a banked system.

37H (55) Output delimiter

CCP initialises this to a 'S' as the delimiter character for strings to the console output. If the character string contains a 'S', then this byte can be changed, typically to a null. .cp5

38H (56) CTRL-P echo to list output flag

The BDOS intercepts CTRL-C from the console to turn on and off the echo of console output (except direct i/o) to the list device. Setting this byte to 01 enables the echo, a value of 00 disables.

3CH (60) Current DMA address

CCP initialises this to 0080H, but as the BDOS maintains two records of the current DMA address this field must never be written to, but may be read to determine the current value.

3EH (62) Current Disk

The current disk is reset to the default disk (0-15) by the CCP. The current disk may be temporarily changed by BDOS FUNCTION 14, or reset to disk A by BDOS FUNCTION 13.

3FH (63) Pointer to FCB in use by BDOS

If byte 41H set to 0FFH, then contains the address of the FCB currently being used by the BDOS.

42H (66) BDOS function number at last disk error

44H (68) Current User number

The current user is reset to the default user (0-15) by the CCP. The current user may be temporarily changed by BDOS FUNCTION 32.

4AH (74) Current Multi sector count

The multi sector count determines the number of 128 byte records (maximum 128) which are transferred on each disk read or write (sequential and random) command. It is initialised to 1 by the CCP.

4BH (75) Disk error mode (00H, FEH or FFH)



- 4CH (76) Drive search chain  
4 bytes defining up to 4 different drives to be searched for a file. Each byte may be set to 0 for the default drive, 1 thru 16 for the drive A thru P or 0FFH if end search. Used by CCP to find file.
- 50H (80) Temporary file drive  
Initialised to 0 for the current drive, but may be set to 1 thru 16 corresponding to drive A-P. Few utilities use this drive. Used by CCP.
- 51H (81) Disk number at last disk error
- 54H (84) Media flag
- 57H (87) Flags: (Bit 7 - message, 6 - alloc)
- 58H (88) Date stamp - days since 1st Jan '78
- 5AH (90) Date stamp - hours (BCD)
- 5BH (91) Date stamp - minutes (BCD)
- 5CH (92) Date stamp - seconds (BCD)  
If the date stamp is accessed directly from this system control block rather than through BDOS FUNCTIONS Set & Get TOD, then the BIOS FUNCTION 26 should be called before any such read, and after any update.
- 5DH (93) Common Memory Base Address  
In a banked system, this determines the start of common memory, and any RSX that needs to remain in memory during a bank switch to the system bank can use this address as a check. A value of zero indicates an unbanked system.
- 5FH (95) Error message jump  
This jump can be intercepted by the BIOS to produce customised error messages.
- 62H (98) Address of current top of User TPA  
This is the value placed in zero page location 6-7 by the BIOS on a warm boot. The loader adjusts this value for any RSXs included below the BDOS.

Not all items available through the BDOS functions are maintained in the system control block, for example the following items are maintained in the banked portion of the BDOS:

Login vector (BDOS 24, 37)  
Read Only vector (BDOS 28 & 29)  
Return Serial Number (BDOS 107)  
Login vector (BDOS 24, 37)  
Read Only vector (BDOS 28 & 29)  
Return Serial Number (BDOS 107)  
Default Password (BDOS 106)

+-----+   BDOS FUNCTION 49 - Get/Set System Control Block   +-----+	
Calling parameters	
Register C = 31H	
Register DE - SCBPB address	
SCB Parameter Block - Get	
Byte 00    - Offset within System Control Block	
Byte 01    = 00H for Get function	
SCB Parameter Block - Set	
Byte 00    - Offset within System Control Block	
Byte 01    = FFH to Set a byte field	
= FEH to Set a word field	
Byte 02    - Byte value to be set	
Byte 02-03 - Word value to be set	
Returned parameters - Get function	
Register A = Returned Byte	
Register HL = Returned Word	
+-----+	

### 7.11. BDOS system and miscellaneous functions

These functions need no introduction and comprise the following BDOS functions:

BDOS FUNCTION 0 - System Reset

BDOS FUNCTION 12 - Return Version Number

BDOS FUNCTION 107 - Return Serial Number

BDOS FUNCTION 108 - Get/Set Program return code

The BDOS system functions

-----

7 - BDOS & BIOS - BDOS System and Miscellaneous Functions

```

+-----+
| BDOS FUNCTION  0 - System Reset |
+-----+
| Calling parameters |
|   Register  C = 00H |
| |
| No Returned parameters |
| |
| The calling program is terminated and control returns to |
| the CCP. |
+-----+

```

```

+-----+
| BDOS FUNCTION 12 - Return Version Number |
+-----+
| Calling parameters |
|   Register  C = 0CH |
| |
| Returned parameters |
|   Register HL = 0031H      (CP/M Plus) |
|                   = 0022H      (CP/M 2.2) |
|                   = 0000H      (CP/M 1.4 and earlier) |
|                   = 0130H      (MP/M II) |
|                   = 0222H      (CP/NET) |
+-----+

```

```

+-----+
| BDOS FUNCTION 107 - Return Serial Number |
+-----+
| Calling parameters |
|   Register  C = 6BH |
|   Register DE - Serial Number field address |
| |
| Returned parameters |
|   Serial Number Field set to 6 byte serial number |
|   Unfortunately few copies of CP/M Plus are actually |
|   serialised - most have 654321. |
+-----+

```

+-----+	
BDOS FUNCTION 108 - Get/Set Program Return Code	
+-----+	
Calling parameters - Get	
Register C = 6CH	
Register DE = 0FFFFH	
Calling parameters - Set	
Register C = 6CH	
Register DE - Program return Code (0000H - FFFEH)	
CCP conditional classification of Program Return Codes	
0000 - FFFF   Successful Return	
FF00 - FFFF   Unsuccessful Return (except FFFE)	
Reserved Program Return Codes	
0000           CCP initialised value	
FF80 - FFFC   Reserved	
FFFD           Program terminated through fatal BDOS error	
FFFE           Program terminated as user typed CTRL-C	
Returned parameters - Get function	
Register HL = Program Return Code	
+-----+	

## 7.11.2 BDOS functions external to BDOS

Not all of the BDOS functions are actually processed by the BDOS. These are special functions for a which a BDOS number is reserved and include:

BDOS FUNCTION	47 - Chain to program	(CCP function)
BDOS FUNCTION	50 - Direct BIOS calls	(BIOS function)
BDOS FUNCTION	59 - LOAD OVERLAY	(LOADER function)
BDOS FUNCTION	60 - Direct RSX calls	(RSX function)

BDOS FUNCTION 47 - Chain to program	(CCP function)
<p>Calling parameters</p> <p>Register C = 2FH</p> <p>Register E - Chain flag</p> <p>0080H-00FFH - Command line terminated with a null</p>	
<p>Chain Flag: = 0FFH to initialise default drive and user to current values.</p> <p>&lt; 0FFH to initialise default drive and user to default CCP values.</p>	
<p>BDOS sets the flag in the system control block offset 17H bit 7 for chain and sets bit 6 to use current values. BDOS then performs a warm boot. The command line must always be placed in the CCP DMA buffer at address 0080H, and not the current DMA buffer (if changed).</p>	
<p>This function is performed by the CCP, thus there are no returned parameters. If any errors occur control continues in the CCP in the usual manner.</p>	
<p>BDOS function 108 may be used to set program return code to pass a 2 byte value to the chained program.</p>	

---

BDOS FUNCTION 50 - DIRECT BIOS CALLS (BIOS function)

---

## Calling parameters

Register C = 32H

Register DE - BIOS PB address

## BIOS PB

Offset Size Description

00 byte BIOS function number (1=Warm Boot)

01 byte A Register contents

02 word BC Register pair contents

04 word DE Register pair contents

06 word HL Register pair contents

This function must be used for all BIOS functions that are not resident. The BIOS jump vector can be used for the character i/o functions.

BIOS function 27 is intercepted without any change in the memory selection.

## Returned parameters

The BDOS intercepts the return parameters and only passes parameters in registers A and HL. The BDOS does not return the address of the DPH in BIOS function 9, instead the BDOS copies the DPH into resident memory and returns the address of this copy of the DPH.

---



BIOS FUNCTIONS - CP/M PLUS		
Function No.	Input	Output
BOOT	0 Do not use	
WBOOT	1* none	none
CONST	2* none	A=FF (Ready), =00 (No char)
CONIN	3* none	A=character
CONOUT	4* C=char	none
LIST	5* C=char	none
AUXOUT	6* C=char	none
AUXIN	7* none	A=character
HOME	8 none	none
SELDISK #	9 C=drive(0-15)	HL=DPH addr, =0000 (no disk)
	E bit 0 = flag	none
SETTRK	10 BC=track	none
SETSEC	11 BC=sector	none
SETDMA	12 BC=DMA address	none
READ	13 none	A=00, 01(error), FF(media)
WRITE	14 C=deblocking code	A=00, 01(error), 02(r/o) =FF(media)
LISTST	15* none	A=FF (Ready), =00 (Busy)
SECTRN	16 BC=logical sector DE=XLATE address	HL=sector
CONOST	17* none	A=FF (Ready), =00 (Busy)
AUXIST	18* none	A=FF (Ready), =00 (No char)
AUXOST	19* none	A=FF (Ready), =00 (Busy)
DEVTBL	20* none	HL=CHRTBL addr
DEVINI	21* C=device(0-12)	none
DRVTL	22 none	HL=DPH table addr
MULTIO	23 C=multisector count	none
FLUSH	24 none	A=00, 01(error), 02(r/o) =FF(media)
MOVE \$	25* HL=destination DE=source BC=length	HL=HL+BC DE=DE+BC
TIME	26* C=00(Get) FF(set)	HL&DE unchanged
SELMEM	27 A=memory bank	none
SELBNK	28 A=DMA memory bank	none
XMOVE	29 B=destination bank C=source bank	none (BC saved for next MOVE function)
USERF	30 - implementation dependent -	
RESERV1	31 - reserved for future use -	
RESERV2	32 - reserved for future use -	
Notes:		
* These functions may be called directly using the Warm Boot address.		
\$ These functions cannot return parameters through BDOS 50		
# These functions copy data blocks into common memory		

BDOS FUNCTION 59 - LOAD OVERLAY	(LOADER function)
<p>Calling parameters</p> <p>Register C = 3BH</p> <p>Register DE - Address of FCB or 0000h</p> <p>FCB R0 &amp; R1 - Load address</p> <p>Top of stack- Address for control after load</p> <p>FCB - Contains an 'opened' FCB with random address bytes R0 and R1 initialised to load address</p> <p>FCB = 0000h is a special option in which no FCB is passed instead all removable RSX are removed</p> <p>This function is not included in the BDOS, instead the LOADER RSX intercepts the call if the LOADER is present. The CCP uses this loader to load COM or PRL files with or without attached RSX's. CCP initialises the top of the stack to 0100H followed by 0000H to transfer control to the loaded program.</p> <p>If the top of stack contains 0100H and no RSX's are present after loading, then the jump vector at 6,7 and at SCB offset 99 are set to the BDOS entry point which removes the loader.</p> <p>IMPORTANT if the address in (SP) is not 0100h or the load address in the FCB R0-R1 is not the same as the address in (SP), then the calling program must ensure that the loaded program does not corrupt the calling program or stack.</p> <p>Returned parameters</p> <p>If (SP) = 0100H</p> <p>Successful load - passes control to 0100h</p> <p>Any error - displays error message and warm boots</p> <p>IF (SP) &gt; 0100H, then returns</p> <p>A = 00H if successful</p> <p>A = 0FEH if bad address or no memory</p> <p>A = 0FFH if physical error and extended errors enabled</p> <p>For a greater insight into this function, refer to APPENDIX E - the disassembly of the CCP.COM file.</p>	

If the BDOS function 59 is to be used to load an RSX, then the COM file must contain a GENCOM header page, a NULL COM page followed by the RSX page and bit map, and the load address must be 0100H.

The GENCOM page consists of:

-----+-----			
GENCOM HEADER			(LOADER function)
-----+-----			
Offset	Size	Description	
-----+-----			
00H	01H	'C9H' Identifies GENCOM	
01H	word	Length of COM program starting at offset 0100H	
03H	0Dh	initialising routine in header. RET terminates	
1st RSX			
10H	word	Offset to start of RSX module 0000 = end of RSX	
12H	word	Length of RSX module (excludes bit map)	
14H	byte	Set to 0FFH if RSX not loaded in banked system	
2nd RSX			
20H	word	Offset to start of RSX module 0000 = end of RSX	
22H	word	Length of RSX module (excludes bit map)	
24H	byte	Set to 0FFH if RSX not loaded in banked system	
etc untill RSX offset = 0000			
100H	01H	'C9H' Identifies NULL COM program and sets	
		System Control Block offset 17H bit 1	
-----+-----			

Offset 3 provides for initialisation and is used in programs such as GET and PUT to change the system control block. The initialisation is called after the RSX's have been loaded, but before the COM file is correctly located at 0100H. The initialisation process must RETURN to the loader program.

BDOS FUNCTION 60 - DIRECT RSX FUNCTION	(RSX function)
Calling parameters	
Register C = 3CH	
Register DE - RSX PB Address	
RSX PB	
Offset Size Description	
00 byte RSX function number	
01 byte Number of word parameters	
02 word Parameter 1	
04 word Parameter 2	
etc. to Parameter n	
<p>This function is not included in the BDOS, instead it is intended for an RSX to intercept the call and process it if the RSX is active for the function number specified. Function numbers 128 to 255 are reserved for system use (Most, if not all, system RSX's do not use this function)</p>	
<p>This call enables RSX's to expand the number of pseudo BDOS functions without interfering with any existing BDOS function numbers.</p>	
Returned parameters	
<p>The returned parameters are determined by the RSX that intercepts the function. If no RSX intercepts the function and it reaches BDOS, then BDOS will return the error code for number out of range (HL=00FFH)</p>	

### 7.12. CP/M 2.2 BDOS compatibility

The CP/M Plus operating system includes many additions and enhancements over the previous version, CP/M 2.2. Fortunately, the changes to the CP/M 2.2 BDOS interface were in general limited to the more obscure functions. All, well almost all, of the standard CP/M 2.2 software provided from the major software companies will run under CP/M Plus. For this we have to thank the earlier MP/M computer systems, as many of the major items of software were found to contain serious flaws when running under MP/M. At the time, it was thought MP/M would become a very popular system, so the software vendors cleaned up their software, removing all those 'tricks' of the trade, to provide software which adhered to the popular CP/M 2.2 interface.

The change between 2.2 and Plus that is most likely to effect the user is the replacement of the IOBYTE with the character device re-direction. Fortunately, as the IOBYTE was only an optional feature of CP/M 2.2 systems, most software did not rely on this feature. Wordstar, for example, allows the IOBYTE to be used to support Printers with software X-ON/X-OFF handshaking. On a CP/M Plus system, Wordstar can quickly be re-installed without this option, and CP/M Plus can then be configured through the DEVICE utility to enable X-ON/X-OFF handshaking.

Any programmer using CP/M Plus will find it difficult to resist using the extra Plus features, so all but the simplest program should include a CP/M version number check and terminate the program unless the CP/M version is 3.1. Alternatively, the program can choose different processing depending on the version.

Whilst the CP/M Plus is largely compatible to CP/M 2.2, there are only a few BDOS functions which have not been affected in one way or another by the Plus enhancements.

### 7.12.1 MEMORY ALLOCATION COMPARISON

CP/M Plus systems generally provide 4k more program space in the TPA area; as shown in the following example:

FFFF			
INTERRUPT VECTORS	.5K	.5K	INTERRUPT VECTORS
BIOS		1K	RESIDENT BIOS
			SCB DATA
	3k	1.5K	RESIDENT BDOS
BDOS			61K TPA
	3.5k		
57K TPA			

## 7.12.2 LOCATION OF CCP (Console Command Processor)

The CP/M Plus CCP exists as a separate CCP.COM program which is loaded and executed in the same way as an ordinary '.COM' file. As such it is loaded at address 0100H. The CCP.COM program then executes in TPA.

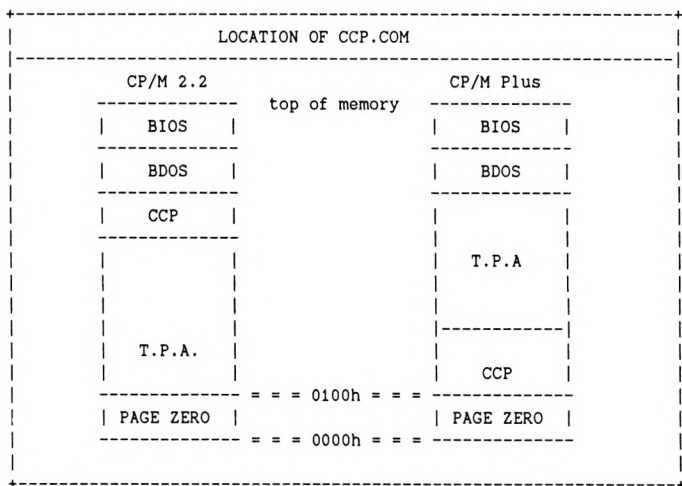
In contrast the CP/M 2.2 CCP is part of the BDOS and loaded into high memory whenever a warm boot occurs. This change in methodology can affect the user in two ways because the CP/M Plus CCP will overwrite any existing program residing in the TPA area.

Consequently:

1. The CCP cannot 'SAVE' the TPA image into a file.
2. A program already in the TPA cannot be re-executed by passing a null file name to the CCP prompt.

Under CP/M 2.2 it is very easy to save a program that is in the TPA onto a disk. This is the only way any program altered by DDT (or SID or ZSID) could be saved back on a disk. Under CP/M Plus, the Plus utilities have been extended to include a Save option, but the use of any 2.2 program which expects the CCP to be able to 'SAVE' a file will cause difficulties. CP/M Plus offers a SAVE RSX facility, but this depends on the user setting this up in advance.

The loading of a null file, is a useful trick under CP/M 2.2. The null '.COM' file is an empty file, in that it has no data records at all. CCP loads this file in the normal way, but because there is no data, nothing is transferred into the TPA. Then CCP passes control to the start of TPA at 0100h and the previous program loaded can be re-executed.



## 7.12.3 ZERO PAGE COMPARISON

Apart from the loss of the IOBYTE, there is very little alteration to the CP/M 2.2 Zero Page specification by CP/M Plus. The Byte at offset 0004h still contains the default drive and user, but it is read-only under CP/M Plus. Under CP/M 2.2, changing this byte also changed the default drive.

ZERO PAGE COMPARISON				
	FUNCTION	O.S. Vers.		Changes to CP/M 2.2
		2.2	Plus	
0000	WARM BOOT	yes	yes	
0001	BIOS entry	yes	yes	
0003	IOBYTE	yes	no	no longer supported
0004	DR/USR	yes	yes	CCP uses value in SCB
0005	BDOS	yes	yes	
0006	BDOS entry	yes	yes	BDOS Fn 59 may change value
0050	DRIVE	no	yes	new feature
0051	PASSWORD 1	no	yes	new feature (address)
0053	PASSWORD 1-L	no	yes	new feature (length)
0054	PASSWORD 2	no	yes	new feature (address)
0056	PASSWORD 2-L	no	yes	new feature (length)
005C	FILE SPEC 1	yes	yes	
006C	FILE SPEC 2	yes	yes	
0080	BUFFER	yes	yes	

## 7.12.4 CP/M 2.2 UTILITY COMPATIBILITY

Most of the utilities supplied with CP/M 2.2 will function under CP/M Plus. The most obvious incompatibilities are:

SYSGEN - Do not attempt to use under CP/M Plus

MOVCPM - Do not attempt to use under CP/M Plus

SUBMIT - Does not work, the CP/M Plus submit is much better. However Plus supports the \$\$\$SUB files created by the CP/M 2.2 Submit or other programs.

XSUB - Does not work. Replaced by CP/M Plus Submit.

STAT - Works ok, except cannot calculate free disk space, nor provide IOBYTE functions.

PIP - Safe to use accidentally, better to use Plus version.

## 7.12.5 COMPATIBILITY OF BDOS FUNCTIONS

With very few exceptions, the CP/M 2.2 BDOS Functions are compatible with the CP/M Plus functions; there are, however, very few BDOS Functions which have not been enhanced by CP/M Plus. The following table summarises the changes.

PLUS BDOS FUNCTIONS	CP/M 2.2 COMPATIBILITY	ADDITIONAL FEATURES
0 RESET	yes	-
1 CONIN	yes	yes
2 CONOT	yes	yes
3 AUXIN	yes	-
4 AUXOT	yes	-
5 LIST	yes	-
6 DIRIO	almost	yes
7 AXINS	NO	yes
8 AXOTS	NO	yes
9 STRING	yes	yes
10 CONBUF	almost	yes
11 CONST	yes	yes
12 VERS	NO	-
13 DSKRES	yes	yes
14 SELECT	yes	yes
15 OPEN	almost	yes
16 CLOSE	almost	yes
17 FIRST	yes	yes
18 NEXT	yes	yes
19 DELETE	yes	yes
20 READ	yes	yes
21 WRITE	yes	yes
22 MAKE	yes	yes
23 RENAME	yes	yes
24 LOGIN	yes	-
25 CURDSK	yes	-
26 SETDMA	yes	yes
27 ALLOC	NO	-
28 WRPROT	yes	-
29 RDVECT	yes	-
30 ATTRIB	yes	yes
31 DPB	NO	-
32 USER	almost	-
33 RNREAD	yes	yes
34 RNWRITE	yes	yes
35 SIZE	almost	yes
36 RANDOM	yes	yes
37 DRRESET	yes	yes
40 ZROFILL	yes	yes



## 7 - BDOS & BIOS

The following functions are not included in CP/M 2.2:

38 ACCESS			
39 DRFREE			
41 TSTWR	48 FLUSH	100 LABEL	107 SERIAL
42 LOCK	49 SCB	101 LABST	108 RETURN
43 UNLOCK	50 BIOS	102 STAMP	109 CNSMODE
44 COUNT	59 OVERLAY	103 XFCB	110 STRDELIM
45 ERROR	60 RSX	104 SETTOD	111 PRTEXT
46 SPACE	98 BLOCKS	105 GETTOD	112 LSTTEXT
47 CHAIN	99 EOF	106 PASSW	152 FILENAME

### 7.12.6 MAJOR INCOMPATIBILITIES OF BDOS FUNCTIONS

#### BDOS FUNCTION 7 & 8

CP/M 2.2 - GET SET IOBYTE  
CP/M 3.1 - AUXILIARY STATUS

#### BDOS FUNCTION 10 (READ CONSOLE BUFFER)

CP/M 2.2 - CONSOLE BUFFER TERMINATED WHEN FULL  
CP/M 3.1 - RINGS BELL WHEN FULL

#### BDOS FUNCTION 27 (GET ADDR (ALLOC))

CP/M 2.2 - ADDRESS RETURNED POINTS TO ALLOC VECTOR  
CP/M 3.1 - ADDRESS RETURNED MAY POINT TO ALLOC VECTOR IN  
DIFFERENT MEMORY BANK

#### BDOS FUNCTION 15 (OPEN FCB)

CP/M 2.2 - FCB MAY CONTAIN WILDCARD  
CP/M 3.1 - WILDCARDS CAUSE EXTENDED ERROR

#### BDOS FUNCTION 16 (CLOSE FCB)

CP/M 2.2 - RECORD COUNT COULD BE REDUCED TO TRUNCATE FILE  
CP/M 3.1 - ANY REDUCTION IN RECORD COUNT IS IGNORED

### 7.12.7 MINOR INCOMPATIBILITIES OF BDOS FUNCTIONS

#### BDOS FUNCTION 1, 2, 9 & 10 (CONSOLE I/O)

CP/M 2.2 - SCROLLING RESTARTED BY MOST CHARACTERS  
CP/M 3.1 - SCROLLING STARTED ONLY BY CTRL-Q

#### BDOS FUNCTION 6 (DIRECT CONSOLE INPUT)

CP/M 2.2 - IGNORE 1 CHAR CONSOLE I/O LOOK AHEAD BUFFER  
CP/M 3.1 - WILL INPUT ANY CHARACTER IN LOOK AHEAD BUFFER

#### BDOS FUNCTION 10 (READ CONSOLE BUFFER)

CP/M 2.2 - PARITY BIT SET TO ZERO  
CP/M 3.1 - 8 BIT CODES ACCEPTED

#### BDOS FUNCTION 13 (RESET DISK SYSTEM)

CP/M 2.2 - DESELECTS ALL DRIVES THEN LOGS IN DRIVE A:  
CP/M 3.1 - DOES NOT LOG IN DRIVE A:

#### BDOS FUNCTION 20 & 21, 33 & 34 (SEQUENTIAL,RANDOM READ & WRITE)

CP/M 2.2 - SETS DRIVE R/O IF DISK CHANGED  
CP/M 3.1 - RETURNS ADDITIONAL ERROR CODES

## 7 - BDOS & BIOS

### BDOS FUNCTION 22 (MAKE FCB)

- CP/M 2.2 - EXTENT ZERO NEED NOT BE OPENED
- CP/M 3.1 - EXTENT ZERO MUST BE OPENED TO SUPPORT DATE STAMP

### BDOS FUNCTION 26 (SET DMA)

- CP/M 2.2 - DMA ALWAYS INITIALISED WHEN PROGRAM LOADED
- CP/M 3.1 - DMA NOT INITIALISED BY FUNCTION 59 - LOAD PROG.

### BDOS FUNCTION 29 (GET R/O VECTOR)

- CP/M 2.2 - DRIVE A: IS ALWAYS LOGGED IN SO BIT 0 IS VALID
- CP/M 3.1 - DRIVE A: MAY NOT BE LOGGED IN

### BDOS FUNCTION 31 (GET ADDR (DPB PARMS))

- CP/M 2.2 - RETURNS ADDRESS OF 15 BYTE DPB
- CP/M 3.1 - SELECTS DRIVE IF NOT LOGGED IN  
RETURNS ADDRESS OF 17 BYTE DPB

### BDOS FUNCTION 32 (GET/SET USER NO)

- CP/M 2.2 - USER NUMBERS 0-31
- CP/M 3.1 - USER NUMBERS 0-15 ONLY

### BDOS FUNCTION 35 & 36 (COMPUTE FILE SIZE, SET RANDOM RECORD)

- CP/M 2.2 - R2 IS ALWAYS 0 EXCEPT FOR R2 = 1, R0 & R1 = 0
- CP/M 3.1 - R0, R1, R2 RANGES FROM 0000 to 40000H

## 7.13. BIOS calling conventions

The BIOS is that part of the operating system which is customised to the specific hardware available. In general, programs should not need to access the BIOS directly as, particularly with CP/M Plus, almost all the functions are available through the BDOS. Direct access to the BIOS is, however, common amongst many of the standard packages because earlier versions of the CP/M BDOS interface were insufficient. Consequently, many programs use direct BIOS calls for character input and output.

Even with CP/M Plus, though, there are some functions which can only be obtained through direct BIOS access. For example, the DEVICE program accesses the character device parameter block held in the BIOS. To facilitate this access, the CP/M Plus BDOS provides function 50 for accessing the BIOS. This is necessary with some functions as part of the BIOS may be hidden from the user's program, residing on a different bank of memory. The BDOS function 50 provides full access to these non-resident or banked functions of the BIOS.

For character I/O, programs do not need to use BDOS function 50, and as one of the justifications for making direct BIOS calls for character I/O is to improve on the performance, the use of BDOS function 50 negates this advantage.

## 7.13.1 THE BIOS JUMP VECTOR

Unlike the BDOS with one entry point, the BIOS does not provide a single entry point, instead the BIOS starts with a table of jump vectors for each of the BIOS functions. The address of the 2nd jump vector (warm boot) is placed in the bytes 0001 and 0002 of Page Zero. By adding to this address, each of the BIOS functions can be accessed. Programs can make direct calls to the BIOS jump vector in 3 ways:

1. Using jump offset to warm boot address

```
conout:          ; send character in <> to console
    ld  de,0009h ; Offset to Console Output
    jp  bios
    ....
bios:
    ld  hl,(0001h) ; warm boot jump vector
    add hl,de
    jp  (hl)
```

2. Using BIOS function number (See BDOS Funtion 50)

```
conout:          ; send character in <> to console
    ld  e,4      ; Bios function number
    jp  bios
    ....
bios:
    ld  d,0
    ld  hl,(0001h) ; function 1 jump vector
    dec de
    add hl,de
    add hl,de
    add hl,de
    jp  (hl)
```

3. Creating BIOS jump table in program

```

        dseg
wboot:  jp  $-$ ; 1
const:  jp  $-$ ; 2
conin:  jp  $-$ ; 3
conout: jp  $-$ ; 4
list:   jp  $-$ ; 5
auxout: jp  $-$ ; 6
auxin:  jp  $-$ ; 7
        ds  7*3
listst: jp  $-$ ; 15
        ds  1*3
const:  jp  $-$ ; 17
auxist: jp  $-$ ; 18
auxost: jp  $-$ ; 19
devtbl: jp  $-$ ; 20
devini: jp  $-$ ; 21
        ds  3*3
move:   jp  $-$ ; 25
time:   jp  $-$ ; 26
selmem: jp  $-$ ; 27

lenbios equ $-wboot

        cseg

        ld  (hl),0001h    ; Jump Vector function 1
        ld  de,wboot
        ld  bc,lenbios
        ldir              ; initialise program jmp table
        .....

```

The 3rd example above also illustrates those BIOS functions which can be accessed by a program directly. Access to all the other functions must use the BDOS function 50 because they may reside in the banked portion of the BIOS.

Any parameters to the above functions are passed in register C, and parameters are returned in registers A for single byte, or HL for word.

The banked functions use additional registers for both passing the parameters and for returned parameters.

## 7.14. BIOS character functions

All the character input and output functions are controlled by the redirection vector which determines which of the physical devices are attached to the logical character device.

BIOS function 20 returns the address of the character table CHRTBLE which defines each of the physical character devices. This CHRTBLE data block contains an 8 byte entry for each physical device, and is terminated by a zero byte after the last entry.

Character Device Table CHRTBLE	
Offset	Field
0000H-0007H	Device 0 Control Block
0008H-000FH	Device 1 Control Block
0010H-0017H	Device 2 Control Block
	etc.
	Device n Control Block
	'00h' (Table terminator)

Character Device Control Block		
Offset	Field	Description
0000H-0005H	5 character ASCII device name	
0006H	Current Mode Byte	
0007H	Current Baud Rate (= 00H if no baud)	

Character Device Mode Bits	
Bit	Meaning if Set
0 (00000001B)	Device can provide input (eg. keyboard)
1 (00000010B)	Device can accept output (eg. printer)
2 (00000100B)	Device has selected baud rates
3 (00001000B)	Device can generate XON/XOFF protocol
4 (00010000B)	XON/XOFF protocol enabled
5 - 7	Undefined
Note: An input/output device such as a modem has both bits 0 and 1 set.	

Character Device Baud Rates (Bits 0-3 only)		
Byte	value	Baud Rate
0	00H	none (not serial I/O device)
1	01H	50
2	02H	75
3	03H	110
4	04H	134.5
5	05H	150
6	06H	300
7	07H	600
8	08H	1,200
9	09H	1,800
10	0AH	2,400
11	0BH	3,600
12	0CH	4,800
13	0DH	7,200
14	0EH	9,600
15	0FH	19,200

Notes: 1. This table does not provide for split baud rates such as required for Prestel (75/1200), nor does it specify the number of stop bits to be used with each baud rate.

2. The high bits 4-7 should be set to zero.

These features are best illustrated by the CP/M Plus utility DEVICE. The bits of the redirection vector in the system control block are in reverse order, with the most significant bit, bit 15, corresponding to physical device 0.

It is usual for only one bit in each redirection vector to be set thereby assigning just one physical device to the logical device, and for each physical device to be attached to only one logical group of devices (The logical group CONSOLE includes the devices CONIN and CONOUT, and the group AUXILIARY for the devices AUXIN and AUXOUT). Not only does the DEVICE utility actually allow more than one physical device to be assigned to a logical device, but it also allows for a physical device to be assigned to more than one logical group. DEVICE sets more than one bit in any logical device redirection vector for multi device redirection. Each bit represents a physical device. If, for example, a redirection vector has more than one bit set for input, a character will be accepted from whichever physical device has its corresponding bit set, and has a character available. Likewise, on a BDOS input-status call, the logical device will have its input status set to 'true' if one or more of the devices whose redirection bit is set returns a physical status of 'true'. For output, the character is sent to all devices represented in the vector by a set bit, and the output status is only set true, when the status of all such devices are true.

BIOS function 21, DEVINI, is used to initialise the baud rate to the baud rate number contained in the device control block for the device whose number is specified in register C. To change a device baud rate, the program should first change the value in the baud rate field of the appropriate entry in the CHRTBL, then call BIOS function 21 to re-configure the device for the new baud rate. Some systems may use this call to reconfigure other parameters in addition to the baud rate.

BIOS character functions				
No.	WBOOT offset	Name	Passed in Register C	Returned as Byte in register A, address in HL
2	0003H	const:		0FFH=ready, 00H=not ready
3	0006H	conin:		character
4	0009H	conout:	character	
5	000CH	list:	character	
6	000FH	auxout:	character	
7	0012H	auxin:		character
15	002AH	listst:		0FFH=ready, 00H=not ready
17	0030H	conost:		0FFH=ready, 00H=not ready
18	0033H	auxist:		0FFH=ready, 00H=not ready
19	0036H	auxost:		0FFH=ready, 00H=not ready
20	0039H	devtbl:		HL=CHRTBL address
21	003CH	devini:	device 0-15	

The BIOS character I/O functions are:

const: Return Input status of Console device  
conin: Read a character from Console device

conost: Return Output status of Console device  
conout: Send a character to Console device

listst: Return Output status of List device(s)  
list: Send a character to List device

auxist: Return Input status of Auxiliary device  
auxin: Read a character from Auxiliary device

auxost: Return Output status of Auxiliary device  
auxout: Send a character to Auxiliary device

## 7.15. BIOS disk functions

The BIOS disk functions should never need to be accessed for any disc data transfer. However, in very special applications, it may be necessary to examine the specific implementation characteristics. Three of the BIOS disk functions provide information which may be of use:

BIOS function 9 - SELDSK - Select the Specified Disk Drive  
also returns the address of the Disk Parameter Header (DPH)

BIOS function 16 - SECTRN - Translate sector number  
can be used to determine the first sector number on a disk.

BIOS function 22 - DRVTLB - Return address of drive table  
provides a table of logical drives actually implemented.

All these functions must be called using BDOS function 50.

BIOS function 9 - SELDSK, passes the drive number in register C with bit 0 of register E set to zero for first time select. On return, HL is set to the DPH if the drive exists, or 0000h if not. As the DPH may be located in the non-resident memory bank, it cannot be accessed directly; however the BDOS function 50 anticipates this problem and copies the DPH into common memory, returning the registers HL pointing to this copy. In calling function 9, the BDOS passes the login vector into the register pair DE rotated to move the bit corresponding to the 'drive-to-select' into bit 0 of E.

DPH - DISK PARAMETER HEADER		
Bytes	Name	Field
00-01	XLT	Address of logical to physical translate table or set to 0000 if none exists. This address is required for the BIOS function SECTRN.
02-10	-0-	Scratch area
11	MF	If supported, set by BIOS door-open interrupt
12-13	DPB	DPB address - see BDOS function 31
14-15	CSV	Check sum vector address - see DPB field CKS
16-17	ALV	Allocation vector address. For banked systems each datablock requires 2 bits.
18-19	DIRBCB	Buffer Control Block Address for Directory LRU buffering.
20-21	DTABCB	Buffer Control Block Address for Data LRU
22-23	HASH	Directory Hashing table address, or 0FFFFh if none. Contains 4 bytes of compacted FCB spec.
24	HBANK	Set to memory bank number of the hash table.
Note: Apart from the DPB (Disk Parameter Block), the data addresses are unlikely to reside in the TPA bank.		



DPB - DISK PARAMETER BLOCK		
Bytes	Name	Field
00-01	SPT	Number of Logical (128 byte) sectors per track
02	BSH	Datablock block shift factor such that Datablock size = $128 * (2 ** BSH)$
03	BLM	Datablock block mask such that Datablock size = $128 * (BLM + 1)$
04	EXM	Extent mask is such that EXM = maximum size of physical FCB / 16k - 1
05-06	DSM	Maximum datablock number of drive where Maximum number of datablocks = DSM + 1
07-08	DRM	Number of the last directory entry where Maximum number of directory entries = DRM + 1
09-10	AL0/1	16 bit pattern of datablock allocation to the directory.
11-12	CKS	The size of the directory check sum vector. Set to 0000H for no checking, or $=(DRM/4)+1$
13-14	OFF	Number of tracks reserved at the beginning. The first datablock starts at track number OFF
15	PSH	Physical record shift factor such that Record (Sector) size = $128 * (2 ** PSH)$
16	PHM	Physical record block mask such that Record (Sector) size = $128 * (PHM + 1)$

BCB - Buffer Control Block Fields		
Bytes	Name	Field
00	DRV	Drive number (0-15), or 0FFH if empty
01-03	REC	Physical Record Number (base 00000H) where Random Record Number = $REC * (2 ** BSH)$
04	WFLG	=0FFH if pending write, otherwise = 00H
05	00	Scratch area
06-07	TRACK	Track number on which REC resides
08-09	SECTOR	Sector number on which REC resides
10-11	BUFFAD	Address of buffer of length $128 * (2 ** BSH)$
12	BANK	Bank number of the buffer
13-14	LINK	Address of next BCB in the linked list. Set to 0000H in last BCB.

BIOS function 16, SECTRN, can be used to determine the number of the first sector on a disk by using BDOS function 50 with the register pairs BC set to zero, and DE filled with the XLT address taken from the DPH of the selected disk. On return SECTRN places the first sector number in register pair HL. This might be useful for very special systems programs which access the disc sectors directly.

BIOS function 22 - DRVTL can be used to determine which logical drives are available for use, or to swap the logical to physical drive assignment. Logical drive swap can be very useful for systems with a Winchester disc to determine which physical drive shall be assigned to drive A. Wordstar, for example, searches for its overlays on drive A, if it cannot find it on the default drive. Swap can arrange for the correct drive to be assigned to logical drive A. See SWAP.COM program at end of chapter.

The BIOS function 22 returns the address of DRVTL in registers HL (using BDOS function 50). In the unlikely event that this address is set to 0FFFFH, or 0FFFEH, then the drive table does not exist.

The DRVTL consists of 16 word addresses corresponding to drives A: to P:. If the address contains 0000H then there is no physical drive allocated. The address is the DPH (disk parameter header) for that drive.

As the DRVTL may reside in bank 0, it can be very difficult to access. The easiest method is for the examining program to be an RSX and therefore residing at the top of the TPA. This program should check to make sure that it is located in common memory. Then by selecting bank 0 using the BIOS function 27 (this cannot be done through BDOS function 50), the DRVTL can be interrogated and the drive allocations swapped if required. The example program SWAP demonstrates bank switching to read DRVTL.

## 7.16. BIOS system functions

Of the remaining BIOS functions, the only ones likely to be of interest are listed in the following table:

BIOS system functions			
No.	offset	Name	Registers passed      Registers returned
1	0000H	wboot:	
25	0048H	move:	HL, DE & BC      HL,DE updated
26	004BH	time:	C=00-Get, =0FFH-Set      HL,DE unchanged
27	004EH	selmem:	A=mem bank      HL,DE,BC unchanged
30	0057H	userf:	- system implementor defined function -

BIOS function 1, WBOOT, is called through location 0000h in page zero. It is the normal method of terminating a program and returning control back to the CCP. The BDOS function 0, also performs a WBOOT.

BIOS function 25, MOVE, is provided primarily for the BDOS to use the instruction LDIR which is only available on the Z80 processor and not by the Intel 8080 & 8085. This BIOS function, after exchanging register pairs HL and DE, can perform a LDIR instruction, exchange registers back and RETURN. If the BIOS is customised for the Intel processor, then the LDIR function must be replaced by additional software.

BIOS function 26, TIME, is provided for those systems which maintain a hardware clock. If the hardware does not support a clock, then this BIOS function simply RETURNS. With a clock, most hardware generates one-second software interrupts, at which time the BDOS TOD clock in the SCB is incremented directly; hence a call to this BIOS function with the Time flag (C) = 00H will result in a return without doing anything in those BIOSs which support interrupts. If the Time flag (C) = 0FFH, then the BIOS must update its clock to the current value in the BDOS SCB TOD field.

BIOS function 27 - SELMEM changes the memory bank to the bank number supplied in register A. Any program which calls this function must reside in common memory with a stack in common memory; consequently the BDOS function 50 does not support this BIOS function. The CP/M Plus system is usually split into two memory banks for the code. These comprise the SYSTEM bank or bank 0 which holds the banked portion of the operating system, and the TPA bank or bank 1. Other banks may be used for the data. This call might be used to access the DRVTLB for example.

BIOS function 30 - USERF is supplied to provide additional functions for the specific hardware and implementation design. For details of the functions provided through USERF, users must refer to the documentation provided with the specific hardware. It is not necessary for USERF to be configured for any functions, or it may be provided for special functions such as formatting the disc which are not normally disclosed to the purchaser of the computer.

## 7.17. Example of Direct Bios Interface

The need to use direct bios calls is clearly demonstrated by the example program SWAP which provides two functions:

1. SWAP - Displays the DRVTL of drive DPH's
2. SWAP d: d: - Swaps logical drive assignments

Because it accesses the DRVTL which may reside in the banked portion of the operating system, direct bios calls are essential to access and change this data table.

The program is in two parts, the SWAPCOM program which initialises the parameters to pass to the RSX, and the SWAPRSX program which makes the direct BIOS calls from common memory. The submit file SWAP.SUB is also included to illustrate the creation of the SWAP.COM file incorporation an RSX.

Program SWAPCOM.MAC

```
.z80
; SWAPCOM.MAC - Combined with SWAPSRX.MAC
; Interrogate DRVTL to display table, or swap entries

bdos      equ      0005h
dfcb1     equ      005ch
dfcb2     equ      006ch
dbuff     equ      0080h

dseg

rsxpb:    ; RSX Parameter Block
          db 127 ; Function number
          db 0   ; Set to 1 parameter for SWAP
          dw 0   ; Space for 2 drives

notplus:  db 'SWAP: ? Requires CP/M 3.1S'
query:    db 'SWAP: ? Bad drives$'

cseg

          ld c,12 ; version
          call bdos
          cp 31H
          ld de,notplus
          ld c,9
          jp nz,bdos

          ld a,(dbuff) ; test command tail
          or a
          jr z,gorsx ; -empty- so just list DRVTL

          ld hl,rsxpb+1
          ld hl,1

          ld a,(dfcb1) ; check 1st drive spec
          dec a
          jp m,error ; -00H- so error
```

```

inc hl
ld (hl),a ; add to RSXPB

ld a,(dfcb2) ; check 1st drive spec
dec a
jp m,error ; -00H- so error
inc hl
ld (hl),a ; add to RSXPB

gorsx: ; NOTE stack is still in CCP LOADER
ld de,rsxpb
ld c,60
jp bdos ; Call RSX & never return

error:
ld de,query ; error in drive spec
ld c,9
jp bdos ; report and terminate

end

```

# Program SWAPRSX.MAC

```

.z80
; SWAPRSX.MAC - Combined with SWAPOOM.MAC
; Interrogate DRVIBL to display table, or swap entries

```

```

cseg

;-----;
; RSX Header ;
;-----;
ds 6
jp drvtbl
bdos:
jp S-S
dw S-S
db 0ffh
db 0
db 'DRVIBL '
db 0,0,0

;-----;
; BDOS SCBPB - Common base address ;
;-----;
scbpbl:
db 5dh
db 00h

memerr:
db '*** Insufficient common memory$'
swapit:
db 'SWAP completed$'

dr1: dw 0 ; 1st drive number (0-15)
dr2: dw 0 ; 2nd drive number (0-15)

```

```

table:                ; copy of DRVTL
    ds    16*2

;-----;
; Start of RSX - Check BDOS & RSX function ;
; NOTE - Stack is assumed to be in CCP    ;
;-----;
drvtbl:
    ld    a,c
    cp    60           ; BDOS function 60
    jr    nz,bdos      ; -not ours-
    ld    a,(de)
    cp    127          ; RSX function 127
    jr    nz,bdos      ; -not ours-
    inc    de
    ld    a,(de)
    cp    1             ; Test for one parameter word
    jr    nz,check     ; -no- so dont fetch any

    inc    de           ; collect 1 parameter word
    ld    a,(de)
    ld    (dr1),a       ; 1st byte is drive 1
    inc    de
    ld    a,(de)
    ld    (dr2),a       ; 2nd byte is drive 2

;-----;
; Check this RSX is loaded in COMMON memory ;
;-----;
check:
    ld    de,schpb1     ; Check above COMMON
    ld    c,49          ; Get SCB
    call    bdos
    ld    de,bdos        ; Use this for check
    ld    a,h
    cp    d
    jr    c,common       ; in common
    jr    nz,cant        ; not in common
    ld    a,e
    cp    1
    jr    nc,common      ; in common
cant:
    ld    de,memerr
    ld    c,9
    jp     bdos          ; display message and return

common:
    ld    hl,(0001)
    ex    de,hl         ; save BIOS warm boot address

    ld    hl,3*(27-1)    ; Select memory bank
    add    hl,de
    push    hl           ; save jmp vector for SELMEM function

;-----;
; CRITICAL CODE AS SWITCHING FROM TPA INTO SYSTEM BANK ;
;-----;

    xor    a            ; system bank

```

```

push de
call jphl      ; BIOS - SELMEM - Select system bank
pop de

ld hl,3*(22-1) ; Return DRV TEL
add hl,de
call jphl      ; BIOS - DRVTEL

push hl
ld de,table
ld bc,16*2
ldir          ; Copy DRVTEL to RSX
pop de        ; recover DRVTEL in DE

ld hl,(dr1)
ld a,(dr2)
cp 1          ; Check for DR1 different to DR2
jr z,list     ; -no- so list DRVTEL

add hl,hl
add hl,de     ; HL -> DPH for DR1
push hl
ld c,(hl)
inc hl
ld b,(hl)     ; BC = DPH for DR1

ld hl,(dr2)
add hl,hl
add hl,de     ; HL -> DPH for DR2
ld a,(hl)
ld (hl),c
ld c,a
inc hl
ld a,(hl)
ld (hl),b     ; DR1 DPH moved to DR2
ld b,a        ; BC = DPH for DR2

pop hl        ; recover -> DPH for DR1
ld (hl),c
inc hl
ld (hl),b     ; DR2 DPH moved to DR1

pop hl        ; jmp to BIOS SELMEM
ld a,1        ; back to TPA bank
call jphl     ; and breath again

ld de,swapt ; SWAP completed
ld c,9
jp bdos      ; display message and terminate
;-----;
; Display DRVTEL ;
;-----;
list:
pop hl        ; jmp to BIOS SELMEM
ld a,1        ; back to TPA bank
call jphl     ; and breath again

```

```

ld de,msg0 ; Header message
ld c,9
call bdos

ld de,table ; Copy in RSX of DRVIBL
ld b,16 ; A: to P: entries
loop:
ld hl,msg1 ; prepare message for each drive
inc (hl)
inc hl
inc hl
inc hl
ld a,(de)
inc de
ld c,a
ld a,(de) ; test DFH = 0000H
or c
ld a,(de)
inc de
jr z,nodph ; -yes-
call hexm ; convert high DFH into HEX
ld a,c
call hexm ; convert low DFH into HEX
jr lst.dr
nodph:
ld (hl),'S' ; no DFH so terminate string early
lst.dr:
push de
push bc
ld de,msg1 ; display created message for drive
ld c,9
call bdos
ld de,crlf ; add new line
ld c,9
call bdos
pop bc
pop de
djnz loop ; repeat for all 16 drives
ret ; terminate RSX

hexm: ; convert number in A
push af ; into HEX ascii in (HL)
rrca
rrca
rrca
rrca
call hex
pop af
hex: and 0fh
add a,90h
daa
adc a,'0'+10h
daa
ld (hl),a ; add ascii HEX character to string
inc hl
ret

cr equ 0dh

```



```

lf equ 0ah

msg0:
db 'DRVTBL:',cr,lf,'$'
msg1:
db '0: 0000H$'
crlf:
db cr,lf,'$'

jphl:
jp (hl)

end

```

Program SWAP.SUB to create SWAP.COM

```

; assemble COM and RSX modules using Microsoft's M80
m80 =swapcom!m80 =swaprxx
; link COM, and RSX as PRL using Digital Research's LINK
link swapcom!link swaprxx[op]
era swap.com!era swap.rxx
; rename files ready for GENCOM
ren swap.com=swapcom.com!ren swap.rxx=swaprxx.prl
; GENCOM combines SWAP.COM and SWAP.RSX into SWAP.COM
gencom swap swap
; SWAP.COM now ready

```

## CHAPTER 8 – RSX extension to operating system

By using RSXs, one can add features or facilities to the operating system. RSXs are very powerful tools, and can, if used carefully, be turned to many interesting applications. Although the RSX is fundamentally designed to extend the facilities of the operating system, one can also implement crude multitasking, such as print-spooling or communications. It can be used to add such things as terminal emulation, an on-screen clock, CP/M 2.2 emulation, mouse facilities, or debugging devices.

As RSXs can be stacked, an effect can be obtained that looks similar to the user as being rather similar to having a number of programs running together.

RSXs are new to CP/M Plus but the ideas are developed from the early days of CP/M. DDT was the first program that relocated itself to the top of the TPA and reduced the size of the TPA accordingly. As DDT predates RSXs and uses the restart vector to gain control of the CPU, it cannot be considered to be an RSX. Neither is DESPOOL, the old CP/M 1.4 print spooling utility, though it is similar. GSX should be, but is not, an RSX. It was actually designed in the days of CP/M 2.2 and a CP/M plus version was never produced. GSX was originally developed by a company (GSS) that was not part of Digital Research.

BDOS calls which are made by a Transient utility must, because they are so revectorized by the loader, pass through any RSXs that are resident at the time. The RSX can intercept any BDOS call or deal with the reserved RSX BDOS call (function 60). RSXs are generally of three types, they can act on BDOS calls to modify the way that they are acted on by the BDOS, they can provide extra BDOS calls, or they can 'steal' CPU time at some or every BDOS call. An example of the first kind might be a patch to insert a temporary AUX device, requiring the redirection of calls 3,4,7 and 8. (Device reconfiguration or redirection would not work!) An example of the second might be a graphics package, or a package to provide access to a sound or music chip. It might also provide access to the joystick. An example of the third kind might be a print spooler.

### 8.1 Structure of an RSX

A RSX program differs from '.COM' program by the inclusion of an RSX Prefix at the start of the program. The prefix is 27 bytes long and most of the RSX Prefix items are initialized by the CCP Loader when the RSX is installed. An RSX should also obey the normal BDOS calling conventions, in that parameters are passed in registers C, D and E, and the parameters are returned in registers A, B, H and L. An RSX should also not use any of the callers stack, instead it should use its own stack. Unlike the BDOS calling conventions of a '.COM' program, the RSX must make its own BDOS function calls through its RSX Prefix and not through location 0005h.

#### 8.1.1 The RSX Prefix

```
.z80
0000 Serial:                ; 6 byte serial number
                        db  0,0,0,0,0,0 ; Initialised by loader
0006 Start:                ; Jump to RSX program
                        jp  rsx_prog
```

## 8 - RSX extension to operating system

```

0009 Next:                ; Jump to Next RSX (BDOS entry point)
      jp  $-$             ; Initialised by loader
000C Prev:                ; Address of Previous RSX, or Page Zero
      dw  7               ; Initialised by Loader
000E Remove:              ; Remove flag
      db  0FFh            ; Set to 0 if the RSX is to be permanent
000F Nonbank:             ; Non banked only flag
      db  0               ; Set to 0FFh for non banked systems only
0010 Name:                ; Name of RSX
      db  '12345678'      ; Any 8-character name
0018 Loader:              ; Loader Flag
      db  0               ; Initialised by loader to 0

0019      ds  2            ; Spare - reserved

```

The following fields of the RSX prefix must be initialised:

```

Start:   Initialise to jump to the RSX program
Next:    The first byte must contain the jump opcode 0C3h
Remove:   Set flag to 00h or 0FFh as required
Nonbank:  Initialise to zero
Name:     Set to name of your choice

```

**START:** This field contains a jump instruction to the entry point in the RSX code. This code should test to see if the BDOS function number passed in register C is to be intercepted, if not, then the code should jump to the NEXT: label without altering the values in registers C, D or E.

**NEXT:** This field is equivalent to the BDOS entry vector at 0005h to be used by the RSX program instead of 0005h. It contains the address of the START: label of the next RSX above it, or to the CCP Loader if this is the top RSX.

**PREV:** This field contains the address of the previous RSX, or if this is the bottom RSX, to the BDOS vector in page zero. When the RSX is first installed, this field is initialised to 0007h. It is modified when another RSX is loaded below it to the offset 0Bh the base of this additional RSX.

**REMOVE:** This flag is usually initialised to 0FFh so that the RSX is removed by the CPP Loader at the next warm boot. Initialise this flag to 00h to make this RSX permanent, it may then be reset to zero when it is no longer required.

**NONBANK:** This flag is always set to zero. It was provided for early CP/M Plus implementations on the dated CP/M 2.2 computers with only 64k of memory. It was used to provide some of the features of CP/M Plus which are excluded in the unbanked version. For example the CP/M Plus utility DIRLBL.RSX contains the BDOS Function 100 for use on unbanked systems.

**NAME:** This 8 character field identifies the RSX for any utility which examines the list of RSX's. The CCP Loader has the name 'LOADER '. Use this field with care as the RSX formed whenever a multiple command line is entered leaves this field uninitialised.

**LOADER:** This flag is always set by the CCP Loader to zero. The CCP Loader itself has this field set to 0FFh. This then flags the end of the chain of RSX's. Caution is needed when scanning for RSX's as this field does not exist in the BDOS so programs must first check if RSX's or the CCP Loader are present by comparing the byte at offset -3 from the start of the system control block which is the page of the start of BDOS, with the byte at location 0007h.

### 8.1.2 The RSX Function calls

An RSX can be used to intercept existing BDOS functions, or it can be used to add new pseudo BDOS functions. CP/M Plus has reserved BDOS function 60 for this purpose. The RSX function number is passed as the first byte in a parameter block addressed by register pair DE in the BDOS call. The RSX function number should be in the range of 0 to 127 and the actual number should be chosen with some imagination. See the BDOS chapter section 7.11.2 for full details and the example shown in section 7.17.

It is advisable to ensure that any RSX which uses this function 60 should conform to the normal BDOS calling conventions and return the result in registers A and/or HL.

### 8.1.3 Creating RSX files

RSX programs are usually created with the GENCOM transient utility which combines RSX's with an (optional) COM file and produces a special COM file starting with a GENCOM header, followed by the original COM (or a NULL COM file if none), and followed by one or more RSX's. See the BDOS Function 59 in section 7.11.2 for details of this header.

Alternatively an RSX can be created directly and an examination of the CCP.COM file at offset 0959H (see Appendix E) demonstrates how this can be done.

### 8.2 RSX Example - spooler

Let us take, as our example, the third type of RSX. This is the most difficult to achieve, and Digital Research's own print spooler RSX was never released in consequence. If an RSX does disc accesses, it can have bizarre effects on the main program in the TPA. If, for example the TPA program does a directory search at the same time that an RSX does a disc access, the BDOS gets terribly confused. If the background task in the RSX changes the DMA address (the area of memory to which disc reads and writes address themselves) while the TPA program is doing a series of reads, the results become rather unhappy. If one wants a happy and trouble-free programming life, then RSXs making disc accesses are to be avoided. We have, in fact taken the bull by the horns and written such a spooler as an example, but we have restricted its allocation of time to buffered console input. (This is easy to change, one just patches the BIOS CONST vector from the RSX.)

Our spooler will send a file, or series of files out through the AUX: port in background.

For a spooler, we want an RSX that is loaded above the top of the TPA, and stays there. To make the example clearer and shorter, we have written the bulk of the RSX in C. This is a foolhardy thing to do and any real RSX should actually be written in assembler. We also need a program that calls the RSX and passes to it the filename of a file that the user wants to be sent via the AUX: logical port. The spooler must maintain the list of files passed to it and must send them sequentially.

Let us start with the easy bit. This transient, called SEND, allows the user to tell the RSX what file to transmit and can be used at any time. It uses the reserved BDOS call 60 to communicate with the spooler RSX. The example could be enhanced, of course, to allow it to send lists of filenames, but we will keep it simple.

```

/* MML:ARM (c) 1985 */
main(argc,argv)
int argc;
char *argv[];
{
    struct rsnb /* as defined by Digital Research */
    {
        /* this is the conventional structure for dialogue
        with RSXs (it can be used for two way data transfer ) */
        char func; /* the function number (we use 5 for our RSX) */
        char numparms; /* not used in this instance */
        char *filename; /* we send a pointer to the filename */
    } our_rsnb;
    if (argc < 2)
    {
        printf("\nSEND is used by supplying the filename as a parameter.\n");
        printf("It is used with the spooling RSX\n");
        exit (0);
    }
    our_rsnb.func = 5;
    our_rsnb.filename = argv[1];
    bdos(60,&our_rsnb); /* make the actual RSX BDOS call no. 60 */
} /* The End */

```

Actually, the program should include such niceties as a version check, but as the authors of this book are not being paid by the word, we will leave out the inessentials.

Now, as we have decided to write the spooler in C, we need an assembly code interface to observe the conventions of the RSX header. In this version, we allow the main program to work only on the first character of buffered line input call (as when you see the A> prompt). It is instructive to alter this to work on a variety of BDOS and BIOS calls. We have chosen the safest course, but you may wait a long time for the file to be sent!

```

;firstly, we have the RSX header. Usually this is of 6 bytes, but
;the linker inserts an extra three bytes as a vector into the main, which
;never gets called
;
    db    0,0,0
    db    0,0,0          ;the loader puts the serial number here
    jmp    entrypoint    ;beginning of the program
next:  db    0C3H          ;a call instruction (8080)
    dw    000             ;jump to the next module in line. If the
;routine makes BDOS calls, then they should be:-
;jmp    next
;
previous:
    dw    000             ;the address of the previous module or 0005H
;if it is the first in line.
    db    00H             ;the remove flag (-1 if it should be removed)

```

;if this is set to 00, then it can set this flag to -1 when it has terminated  
 ;so as to remove the RSX.

```

    db      00H          ;nonbank
    db      'SPOOL      ',0,0,0
                                ;this is the RSX identity string

```

;end of RSX header

```

                                ds      80
our_sp:                        ;our stack
                                ds      4
his_sp:                        dw      00
surpressed:                   db      00
stopped:                      db      00
main ::                        ;dummy entrypoint (never called)
entrypoint:
    push    psw
    lda     surpressed
    ana     a
    jnz     do_next ;is there a lock on this?
    push    b
    push    d
    push    h        ;save the environment
    lxi     h,0
    dad     sp        ;and the stack
    shld    his_sp
    lxi     sp,our_sp
    push    b        ;push the function number
    push    d        ;and the parameter
    xra     a
    dcr     a
    sta     surpressed ;prevent reentrancy
    mov     a,c      ;look at the function number again
    cpi     60       ;is it our special one?
    jnz     not_ours
    call    rsx_entry##
    jmp     hop_over

not_ours:
    cpi     10       ;was it a read console buffer?
    jnz     hop_over

loop_again:
    call    rsx_entry##
    mvi     c,6      ;direct console io
    mvi     e,0feh   ;console status
    call    next     ;from the bdos
    ana     a
    jz      loop_again

hop_over:
    pop     d
    pop     b
    xra     a
    sta     surpressed ;allow RSX_entry to be called
    lhld    his_sp   ;restore the stack
    sphl
    pop     h
    pop     d
    pop     b

do_next:
    pop     psw

```



```

int bc;
{
/* firstly look at the contents of the C register to see if it is the BDOS
call no. 60 */
if ((bc & 0x00FF) == 60)      /* then it may be a 'wake up' request */
{ /* so service the special RSX call */
/* now, is it our call or another one? */
if (de->func != 5) return(); /* if it is not ours then go home */
/* firstly, see if there is a slot in the queue for the filename */
for (i=0; i<QUEUE_SIZE; i++) if (!(the_queue[i].active)) break;
if (i==QUEUE_SIZE) return (FALSE); /* return false if no slot */
strcpy (the_queue[i].filename,de->filename);
the_queue[i].active = TRUE; /* put the file in the queue */
wokenup=TRUE;
return(TRUE);
}
else /* then it was just another bdos call */
{
if (!(wokenup)) return(); /* nothing to do */
/* I wonder if the AUK: port can accept a character? */
if (!(bdos(8,0))) return(); /* if not ready to transmit, do not */
/* if we got here, the auxout is ready */
if (file open)
{
char ch;
if (index < top) /* is there more in the buffer? */
{
bdos(4,ch=buffer[index++]); /* send the character */
if (ch == 0x1A) index=top; /* if EOF, flag it */
}
else /* read in a new buffer full */
{
if (!(top=read (fd,buffer,BUFFER_SIZE)))
{
/* so the file has all been read */
close(fd);
file_open=FALSE;
index=0;
return();
}
else index=0;
}
}
else /* the file was not open */
{
/* get the next file in the queue */
for (i=0; i<QUEUE_SIZE; i++) if (the_queue[i].active) break;
if (i==QUEUE_SIZE)
{
/* there is nothing left in the queue */
wokenup = FALSE; /* we have done all we were woken to do */
return (); /* return as no file to do */
}
else
{
/* we have a file to work on */
the_queue[i].active=FALSE;
fd=open(the_queue[i].filename,O_RDONLY);

```



```

        file_open = (!(fd==1)); /*if no error then file open*/
    )
    ) /* end of else file was not open */
    ) /* end just another BIOS call */
) /* the end of the program */

```

Now, we have the problem of glueing the whole lot together so as to make an RSX. The linker has to be told to produce a '.PRL' file and the GENCOM utility must be used to implant a 'RET' statement as the first byte of the '.COM' file. The easiest way is to construct a '.SUB' file to do it. We have our C compiler in drive A: and we are persuading it to produce an assembly file of RMAC format.

```

a:
cc b:dospool -r
b:
RMAC dospool Spk
RMAC spooler Spk
link spooler[op],dospool,a:libc.rel[s]
era b:spooler.rsx
ren spooler.rsc=spooler.prl
era b:spooler.com
gencom spooler [null]

```

Naturally, this will require modification for other C compilers and other disc configurations.

When all is safely gathered in, and compiled, we need only to type:-

**SPOOLER**

which puts the RSX in place. Do not do this twice, as this version is not protected against multiple installation. One would need to scan subsequent RSXs to find one with the same identity string, before setting the remove flag to zero.

Now typing:-

**SEND filename.typ**

will activate the spooler to send out the file whenever the system is idle. Obviously, to make the program more useful, it would have to steal CPU time at the CONST BIOS call by diverting the BIOS vector. The SETDMA calls would need to be monitored, so that after any change in DMA by the spooler, it can be restored for the TPA program. All SEARCHFIRST and SEARCHNEXT calls would need to put a lock on the system, to be taken off only by another disc call. (eg OPEN CLOSE READ WRITE etc.). When all this is done, then it begins to work rather better, but the listing begins to get too long to make a good example.

## CHAPTER 9 – GSX, The Graphics extension to CP/M

The Amstrad 6128/8256 GSX

### 9.1 Introduction

The average user of the Amstrad 6128/8256 system will, hopefully, never be consciously aware that he has GSX installed on the system. He is aware only that programs written using GSX will run on his computer and allow use of some of the graphics facilities of the Amstrad 6128/8256. Even the programmer need not inquire very deeply into the inner workings of GSX, as he will probably use a programming language such as the CBASIC compiler or Propascal, whose graphics commands look after the interface into GSX. All users should notice that, at last, microcomputer graphics are doing something useful. The fainthearted should, therefore, not read any more than the introduction to this chapter, and hold in his mind the vague impression that GSX is 'a good thing'.

GSX is a standard graphics interface between the program and the specific hardware being used. It attempts to do for graphics what CP/M does for floppy disks. GSX is intended to provide the same general facility for graphics devices that is provided by a disk operating system such as CP/M. In other words, GSX is designed as an interface between the graphics software and the graphical hardware such that every device looks identical to the software. The changing of the output from one device, such as a graphics terminal, to another, such as a plotter, should be as simple as changing one number in a software routine; and the same graph should be representable on every device which has been installed for GSX. GSX represents an implementation of the provisional ANSI GKS standard, which is a standard by which graphics devices are controlled in high-level software.

Obviously, the analogy between a graphical system and a disc-operating system is a bit strained; there are real differences between mass-storage devices and graphics output devices. The quality of output varies enormously between a cheap dot-matrix printer and a laser printer or a plotter. However, the idea of GSX works quite well in practice, despite the limitations of the processing power of microcomputers. GSX was originally written for expensive hardware, and assumes a high-definition screen (or plotter). There are therefore inevitable compromises, and GSX must be judged fairly in this context.

Programs that are written using GSX are much more likely to run on other CP/M machines that also have installed GSX. As the differences between GSX-86, GEM and GSX-80 are minor, the transport of graphics applications programs written in high-level languages between 8 and 16 bit systems is fairly trivial.

Programs that are written specifically for Amstrad 6128/8256 need not use GSX, unless they require the more sophisticated facilities, or need to use a variety of devices, but programs with any pretensions to being standard CP/M packages will need to run their graphics through GSX.

Note that GSX is not particularly suited to fast interactive graphics or three-dimensional graphics work. It is excellent for producing graphs, plots, charts, diagrams, and plans, as well as Visual shells to applications programs. Do not attempt to use it for games unless they are of a sedentary nature.

Programs written using GSX will

- be able to use a large number of printers and plotters
- Be portable to other CP/M systems
- Be able to make use of specific device attributes
- Have access to the sophisticated GSX graphics commands

A CP/M Microcomputer with GSX installed will

- facilitate graphics program development
- Run other graphics programs written on other CP/M micros

to the programmer, GSX provides a graphics operating system that is essentially a part of CP/M. It enables him to write software that is portable to other CP/M systems.

To the end user, GSX allows the running of the new range of Digital Research languages that support graphics, and the commercial graphics programs written for GSX.

GSX is not easy to use directly. The original intention was to use GSX via the package 'GSS-KERNEL' and 'GSS-PLOT'. Unfortunately, the packages suffered from lack of development and, when the agreement between GSS and Digital Research terminated, they were withdrawn, leaving only GSS-GRAPH and GSS-DRAW. These are application packages and are not programmers tools like PLOT and KERNEL. GSS(DR) Graph and GSS(DR) Draw both require GSX and will run on the Amstrad CP/M+ machines. We mention DR Graph in chapter 2. It is an immensely useful program that, by itself, pays for the hardware in terms of the time it saves. DR Draw is of strictly minority interest and would tempt few draughtsmen away from their pencil and paper. GSS-PLOT is a tool to provide graph-drawing extensions to a range of compiled languages whereas GSS-Kernel was intended to provide the GKS interface to them to facilitate a wide range of graphics tasks. If these two packages had been better finished, and written in assembler rather than RatFor, they might have revolutionised the way microcomputers handle graphics.

There is another problem with GSX. It is slow. This is not really the fault of the GIOS (the hardware interface) but is due to the conversion of the data by GSX from Normalised Device Coordinates (NDCs) to actual device coordinates. The ideas of GSX were developed on fast hardware and the design of the software made no compromises for the slow 8-bit processors of the time. It is only with the advent of high-speed 16-bit processors and maths chips that GSX has shown its true worth. Despite the problems with performance, GSX is ideal for applications where slow speed of drawing is no drawback. It is, for example, ideal for business graphics, C.A.D. (computer-aided design) and computer typesetting. A third problem with GSX is that it takes up a significant proportion of the address space of an 8-bit processor. Sadly, there is still no version of GSX that operates in a banked environment like CP/M+ and so there is no way of avoiding cramping the transient program area. On 16-bit machines, this is no problem, and it would have taken a lot of imagination to predict that D.R.I. would have sold nearly a million copies of CP/M+ and GSX in the year after Digital Research virtually abandoned 8-bit technology for the bright lights of the 16-bit processors. Had there been more foresight at Digital Research, we could be enjoying a banked version of GSX, allowing much larger graphics programs to operate on 8-bit CP/M+ machines. As soon as fast maths chips become economic to install on home computers, and the speed of the CPU increases, the true worth of the GKS standard will be better appreciated.

What are the advantages of GSX? Simply, the presence of a collection of device drivers. This means that any GSX-based program can produce graphics output on any device for which a device-driver has been written. Once a new graphics output device has had its GIOS graphics device driver written for it, it will work with a GSX-based program. As GSX is still promoted and maintained by IBM and GSS, there is a clear upgrade path to the faster and more modern processors. Digital Research have an upgraded version of GSX called GEM which is much more complex, but there is a committed support for a GKS-based graphics interface from more than one company. The way that GSX loads its device drivers, keeping only the current one in memory and overlaying the previous ones, is useful in circumstances that demand output to a whole range of output devices. Even 16-bit computers would look cramped if all drivers had to be kept in memory. As it is, one can do surprisingly complex things on an 8-bit micro by simply overlaying drivers one on top of another, just using one at a time.

GSX is an extension to the operating system. Because GSX takes significant TPA space, it does not stay permanently in place. It is placed under the resident BDOS above the TPA only when needed. Any program requiring graphics takes the system from the disk and installs it, in much the same way as a temporary RSX, and it is subsequently jettisoned when the next program is loaded. In this way, it never remains in situ when not needed, taking up valuable TPA space when it is not needed by the program. When a program is created to run with GSX, a copy of the GSX loader is grafted onto the front of the file by the program GENGRAF.COM. When the program is loaded, the loader, rather than the program itself, is executed. This loader relocates GSX to the top of the TPA and activates it so that it examines BDOS calls to detect GDOS calls. It works in a rather similar manner to the RSX of CP/M+. The graphics application program is then moved down to cover the loader and relocate it into its correct place. It is then executed. Even with the GSX system installed, the Amstrad 6128/8256 has over 50K of TPA space available. If GSX is used with one of the supplied GSX hardcopy drivers, then it will take slightly more space.

Obviously, the GDOS, like the BDOS, knows nothing about the actual hardware. The work of implementing graphics commands on the hardware is the responsibility of the GIOS. Every device has its own GIOS. Only one GIOS is resident at any particular time. If a calling program requests a driver that is not yet resident, its driver is taken off the disk and the previous one is overwritten. In this way, one program can appear to drive a number of graphics devices, such as plotters, dot-matrix printers, Daisy wheel printers, etc.

The GDOS, the heart of GSX, is responsible for loading the device drivers requested by the program using GSX, and converting normalized coordinates into hardware device coordinates. The assembly-language programmer communicates with GDOS through a BDOS call 115. As Graphics commands usually need to pass rather more information to the graphics driver than normal BDOS calls, the assembly language programmer needs to pass parameters in a slightly different way to the other BDOS calls.

The Amstrad CP/M+ machines have a GSX driver (or GIOS) which has the minimal required opcodes for a CRT device. It omits all the GIN (graphics input) functions, so it cannot be used as a graphics workstation. Its implementation is 'minimal' in that it allows no change in size of markers, and does not support changes in font, rotation of text, or changes in character size. Moreover, it does not support polygon fill, so that one cannot use it to

display graphs on screen that normally use crosshatching of bars or pie-slices. Later versions of the implementation exist with more opcodes implemented, and are distributed by Digital Research with their DR Graph, DR Draw and CBASIC packages. Before doing any serious work with GSX, it is important to make sure you have one of these updated versions.

GSX comes with drivers for:--

- 1/ Epson FX low resolution (7 bit)
- 2/ Epson FX low resolution (8 bit)
- 3/ Epson FX High resolution (8 bit)
- 4/Anadex DP-9501 and DP-9001A
- 5/ Centronics 351, 352, and 353
- 6/ C.ITOH 8501A low resolution
- 7/ Datasouth DS 180
- 8/ DEC LA50 and LA100
- 9/ IDS Monochrome
- 10/ Okidata Microliner
- 11/Printronic MVP

2/ HP 7470A Graphics plotter

as well as a default driver for Amstrad 6128/8256

On the Amstrad discs are two extra drivers.

- 1/ DD-DMP1.PRL for the DMP1
- 2/ DDSHINWA.PRL for any printer with the SHINWA mechanism

but only the HP 7470A driver in the first list is provided due to lack of space on the disc.

## 9.2 The Amstrad 6128/8256 GSX Formal Device Specification

This describes the implementation of GSX on the Amstrad CP/M plus machines and should be referred-to in order to ascertain any graphic characteristics and limitations that are specific to the implementation.

### 9.2.1 Filenames:

All device drivers will only run under CP/M 3

DDSCREEN.PRL	Device driver for the PCW 8256
DDMODE0.PRL	Device driver for mode 0. 6128
DDMODE1.PRL	Device driver for mode 1. 6128
DDMODE2.PRL	Device driver for mode 2. 6128

### 9.2.2 Device Index:

This is the default device and is given the device index of 01. This device number can be changed by altering the number in the file ASSIGN.SYS

### 9.2.3 Graphics Input:

Not supported!

#### 9.2.4 Text:

Text can only be written in one size and direction. There is only one resident font.

#### 9.2.5 Markers:

There are five marker types

no 1- .	no 2- +	No 3- *	No 4- o
No 5- X			

#### 9.2.6 Linestyles:

There are five hardware linestyles

1-Solid	2-Dashed	3-Dotted	4-Dashed/dotted
5-Grey			

#### 9.2.7 Fill Styles:

No fill styles are supported.

#### 9.2.8 generalized

##### Drawing

##### Primitives:

One drawing primitives are available, Bars.

#### 9.2.9 Escape

##### Functions:

Escape functions available include the following

- 1/ Inquire addressable character cells
- 2/ Exit Graphics mode
- 3/ Enter Graphics Mode  
(The only effect of this function is to turn the alpha cursor off.)
- 4/ Cursor Up
- 5/ Cursor Down
- 6/ Cursor Right
- 7/ Cursor Left
- 8/ Home Cursor
- 9/ Erase to end of screen
- 10/ Erase to end of line  
Erasure is with colour index 0. (6128)
- 11/ Direct cursor address
- 12/ Output Cursor Addressable Text  
(Outputting text in the right hand column will wrap the cursor on to the next line. Outputting text in the bottom right hand corner will cause the screen to scroll.)
- 13/ Reverse Video on
- 14/ Reverse Video off
- 18/ Place cursor at location  
(The graphic cursor is cross consisting of two lines, one horizontal, one vertical, drawn in XOR mode. Thus the cross has a hole in the middle. Each line is 15 pixels long and is drawn using colour index 1.)
- 19/ Remove cursor  
(The cursor is removed by redrawing it in XOR mode. If the cursor has already been removed by other means then this function will redraw the cursor! Once this

function has removed a cursor it has no further effect until another cursor is drawn using PLACE CURSOR AT LOCATION, opcode 5 function 18.)

The ESCAPE functions for finding the cursor address(15), getting the Tablet status(16), and doing a hardcopy (17) are not implemented.

#### 9.2.10 Summary: The following are the implemented functions:

opcode 1            -initialize the graphics screen

The device driver uses the screen in whatever state it finds it, for example in 24 x 80 mode or with the status line disabled.

The graphics part of the GIOS must be run in common memory, 0C000H and above. This should not cause any problems since GSX relocates the GIOS as far up store as it can. Only if there were RSXs filling the top of store would the GIOS be unable to run.

If required the screen may be used in 24 x 80 mode. This should be selected before running GSX.

If enabled the status line may still be used, the device driver will avoid the status line. If disabled the whole screen is used by the device driver. In this case the user is recommended to redirect the CONOUT: device away from the CRT device, otherwise BIOS error messages may appear on the screen spoiling any graphics etc.

If the environment is unsuitable: not CP/M 3, or not a PCW8256, or graphics part of the GIOS not in common memory then the message "Cannot run in this environment" is displayed, the system is then warm booted.

opcode 2            -Stop graphics output to the screen  
                    The screen is restored to its  
                    standard power up state.

opcode 3            -Clear the screen

opcode 4            -Display all pending graphics  
                    This opcode has no effect on the  
                    PCW8256 as the screen is kept up to  
                    date at all times.

opcode 5            -Escape (see above) device-dependent ops

opcode 6            -Output a polyline

opcode 7        -Output markers  
 opcode 8        -Output graphic text  
                  Characters are output using colour  
                  index 0 as pen and the current text  
                  colour attribute as paper.  
 opcode 9        -Display and fill a polygon  
                  The area is only outlined using the  
                  current fill colour index but the  
                  area is not filled.  
 opcode 10       -Define a cell array  
                  The area is outlined using the  
                  current line colour index.  
 opcode 11       -Display a drawing primitive  
                  bar only. The area is outlined using  
                  the current fill colour index.  
 opcode 12       -Set the text size  
                  Only one character height is  
                  supported. The cell is 8 x 8 pixels  
                  and the standard character W is 7 x  
                  7 pixels as follows:

```

10000010
10000010
10000010
10010010
10101010
11000110
10000010
00000000
  
```

opcode 13       -Set text direction. not implemented.  
 opcode 14       -define colour representation  
                  As PCW8256 has a monochromatic  
                  screen only two colour indices and  
                  two colours are available, 0 and 1,  
                  black and white. Any percentage of  
                  colour gives white. On the 6128,  
                  there are three levels of colour  
                  representation are available they  
                  are mapped onto the tenths of  
                  percent units as follows:

```

0           : 0
1.. 500    : 1
501..1000  : 2
  
```

The number of colour indices  
 available in each mode is as  
 follows:

```

mode 0      2 colour indices
mode 1      4 colour indices
mode 2     16 colour indices
  
```

opcode 15       -Set the linestyle for lines

Five line types are supported each  
 is defined by an 8 bit line mask as



follows:

	1	solid	11111111
	2	dash	11110000
	3	dot	11000000
	4	dash dot	11110110
	5	long dash	11111100
opcode 16	-Set	polyline linewidth. not implemented.	
opcode 17	-Set	colour for lines	
opcode 18	-Set	the marker type	
		Five polymarkers are supported:	
		Polymarkers 2..5 are characters	
		positioned centrally on the given	
		coordinates as follows, . marks the	
		marker's coordinate:	
	Marker 2	Marker 3	Marker 4
	+	*	o
	x		
	00000000	00000000	00000000
	00011000	01100110	00000000
	00011000	00111100	00111100
	011.1110	111.1111	01100110
	00011000	00111100	011.0110
	00011000	01100110	01100110
	00000000	00000000	00111100
	00000000	00000000	00000000
opcode 19	-Set	polymarker scale. not implemented.	
opcode 20	-Set	colour for markers	
opcode 21	-Set	Hardware text font. not implemented.	
opcode 22	-Set	colour of text	
		On the PCW 8256 The text colour	
		index only used for graphics mode.	
		Alpha mode always uses the terminal	
		emulator. On the CPC 6128, the text	
		colour index is used for both	
		graphics mode and alpha mode text.	
opcode 23	-Set	interior style of fill	
opcode 24	-Set	fill style index. not implemented.	
opcode 25	-Set	colour for polygon fill	
opcode 26	-Inquire	colour representation not implemented.	
opcode 27	-Inquire	cell array. not implemented.	
opcode 28	-Return	locator position. not implemented.	
opcode 29	-Return	valuator device value. not implemented.	
opcode 30	-Input	choice. not implemented.	
opcode 31	-Input	string. not implemented.	
opcode 32	-Set	current writing mode	
opcode 33	-Set	input mode. not implemented.	

The following GIOS calls are not implemented on the Amstrad machines

#### GENERALIZED DRAWING PRIMITIVE:

ARC	- Opcode 11 Function 2
PIE SLICE	- Opcode 11 Function 3
CIRCLE	- Opcode 11 Function 4

```

PRINT GRAPHIC - Opcode 11 Function 5
SET CHARACTER UP VECTOR - Opcode 13
SET POLYLINE LINE WIDTH - Opcode 16
SET POLYMARKER SCALE - Opcode 19
SET TEXT FONT - Opcode 21
SET FILL INTERIOR STYLE - Opcode 23
SET FILL STYLE - Opcode 24
INQUIRE CELL ARRAY - Opcode 27
INPUT LOCATOR - Opcode 28
INPUT VALUATOR - Opcode 29
INPUT CHOICE - Opcode 30
INPUT STRING - Opcode 31
SET INPUT MODE - Opcode 33

```

The following opcodes are only partially implemented.

```

FILLED AREA - Opcode 9
    This should display and fill a polygon. The polygon is merely
    outlined, using the current fill colour index and is not filled.
    It should be filled in the current colour, with the specified
    'fill' style (eg solid, hollow, pattern or hatch.) in the
    currently specified style. (if hatch).
CELL ARRAY - Opcode 10
    The cell array is not displayed. Instead, its area is outlined
    using the current line colour index.
GENERALIZED DRAWING PRIMITIVE - Opcode 11
    The only GDP implemented is the bar. This is not filled but,
    instead, is outlined using the current fill colour index.
SET CHARACTER HEIGHT - Opcode 12
    As there is only one character size, this does nothing.
SET FILL INTERIOR STYLE - Opcode 23
    As there are no interior styles, this does nothing.
SET FILL COLOUR INDEX - Opcode 25
    As there are no interior styles, this does nothing.

```

Although the Amstrad GSX conforms with the specification of the required subfunctions for a CRT device, it does not implement the means to fill shapes, manipulate the size, direction, or style of text. It cannot allow the user to specify a location on the screen. Only part of DR Graph will work on this, and its functionality is severely restricted. Digital Research therefore supply an upgraded GIOS with their products.

### 9.3 Using GSX.

To use GSX, the user must set up his ASSIGN.SYS file to specify the actual hardware that is available. All drivers must be listed in the ASSIGN.SYS file if they are to be used.

The assignment table in the file ASSIGN.SYS comprises text only and can be modified with any text editor. The ASSIGN.SYS file should be on the currently logged drive. Each entry refers to a GIOS device driver and is accompanied by a device driver no. The convention for driver numbers is as follows:

1-10 = CRTs  
11-20 = Plotters  
21-30 = Printers  
31-40 = GKS Metafile  
41-50 = Other devices

Each entry contains a legal unambiguous filename that corresponds to the name of the device driver on the disk. Unlike CP/M filenames, the ASSIGN.SYS filename must have a drive specifier and if you wish to request GSX to search the default drive, then the special drive specifier @: must be used. The reason for this is obscure.

for example, the average Amstrad 6128/8256 user might just require the DDSCREEN driver in which case the assign.sys file might contain

```
1 @:ddscreen
```

and nothing more. (if the .prl file was on M: drive then the response would be faster. the entry would be 01 M:DDSCREEN )

When using the High resolution FX driver for good quality graphs, then the following might be used

```
21 @:ddfxhr8  
01 @:ddscreen
```

Note that the GDOS allocates as much or as little space as is required to load the first GSX driver in the list and therefore the first in the list must be the biggest driver. The application package should not assume that the system console is loaded in by default but should consciously load the CRT workstation.

A compiled program that has been written to work under GSX needs to have the GSX loader fastened on to it with the GENGRAF.COM utility.

## 9.4 Programming with GSX

Normally, the programmer will not involve himself directly with the GDOS but will use KERNEL or PLOT to interface his high-level language with GSX or he will use the graphics interface into GSX embedded into the language itself.

In the newer versions of CBASIC, for example, there are functions to access GSX.

## A) Control functions

```

GRAPHIC OPEN N
GRAPHIC CLOSE
SET BEAM "ON"
SET BEAM "OFF"
ASK BEAM STRS
SET CLIP "ON"
SET CLIP "OFF"
SET BOUNDS X.VAL, Y.VAL
ASK BOUNDS X.VAR, Y.VAR
ASK DEVICE X.VAR, Y.VAR
SET VIEWPOINT X1, X2, Y1, Y2
ASK VIEWPOINT X1, X2, Y1, Y2
SET WINDOW X1, X2, Y1, Y2
ASK WINDOW X1, X2, Y1, Y2

```

## B) Mode functions

```

SET COLOR COUNT C.VAL
ASK COLOR C.VAR
SET COLOUR C.VAL

```

## C) Line Drawing Functions

```

MAT PLOT N: X.ARRAY, Y.ARRAY
PLOT (X1, Y1), (X2, Y2), (X3, Y3), (X4, Y4), (X5, Y5).....etc
SET LINE STYLE LS.VAL
ASK LINE STYLE LS.VAR

```

## D) Marker Drawing Functions

```

MAT MARKER N: X.ARRAY, Y.ARRAY
SET MARKER HEIGHT Y.VAL
ASK MARKER HEIGHT Y.VAR
SET MARKER TYPE N

```

## E) Filled Area Functions

```

MAT FILL N : X.ARRAY, Y.ARRAY

```

## F) Graphic Text Functions

```

GRAPHIC PRINT AT (X,Y) : "text string"
SET CHARACTER HEIGHT Y.VAL
ASK CHARACTER HEIGHT CH.VAR
SET TEXT ANGLE THETA.VAL

```

```
ASK TEXT ANGLE TA.VAR
SET JUSTIFY X.VAL, Y.VAL
ASK JUSTIFY X.VAR, Y.VAR
```

G) Alpha and Graphic cursor and Alpha Text Functions

```
SET POSITION X.VAL, Y.VAL
ASK POSITION X.VAR, Y.VAR
CLEAR
```

I) Graphic input functions

```
GRAPHIC INPUT X.VAR, Y.VAR, CHARS
```

Any programmer intending to do serious work with GSX should consult the two GSX manuals and the documents and programming examples distributed by DR. We include a section (9.5) that details all the GDOS calls, thereby providing enough to get started with. Unless one is prepared for some considerable effort; it is much better to use the CBASIC compiler (CB80) with graphics extensions for writing GSX-based programs. If determined to proceed with a low-level interface into GSX then the following must be noted:

- Coordinates passed to GSX are normalized Device coordinates (NDCs) that go from 0-32767 along each axis.
- Commands can be divided into:-
  - 1/ Workstation controls (initialization, clearing, closing etc)
  - 2/ Output primitives ( Polyline, polyfill, polymarker, text etc)
  - 3/ Primitive attribute controls (colour, linetype, text direction )
  - 4/ Inquiry functions
  - 5/ Graphics input functions
  - 6/ 'Escape' sequences for screen and graphic cursor functions
- All information to and from the GDOS is passed as 16 bit integers
- All integers are passed as arrays
- All array indices are 1 relative (ie go from 1...n rather than 0...n-1)
- All coordinate information is in the NDC ( Normalized Device coordinate) space

Having determined to use GSX, the first task is to design and program an interface into your favourite high-level language. We will mainly use C here, as it is the most popular language for microcomputers currently available, and we want our programs to run in a 16-bit environment as well as in the 8-bit don't we? Strangely, the most complex task is to open a workstation (which means to select and activate a particular graphic device). Once we have achieved this, the whole process becomes one of having the manual (or this book) in one hand, the keyboard in the other and a clear head on your shoulders.

The effect of opening a workstation is that GSX, which has wormed its way into the operating system, searches in a file on disc called ASSIGN.SYS for the name of a device driver file corresponding to the device driver number that you specify. If, for the sake of example, you wish to use the graphics console (device 01), you open the workstation specifying the device number 01. GSX searches the file ASSIGN.SYS for the filename corresponding to 00 in a list of files. If it finds it, DDWHIZZO for example, it will look for the device driver DDWHIZZO.PRL and load it into the area which it reserves for the GIOS (Graphics

input/output system). Having done all this, it will then call the GIOS 'open\_workstation' command. This does whatever is necessary, such as selecting graphics mode, initialising the device, blanking the screen and the like, and passes back to GSX, and the calling program, a great variety of data that describes the device, its capabilities and its characteristics. It is important to remember that there is enormous variety in graphics device, ranging from dumb output from a dot-matrix printer to both input and output from a colour graphics workstation with keyboards, mice, trackballs, digitisers and joysticks. One really needs to know what sort of beast one is dealing with.

To call GSX, it is necessary to call the BDOS using function code 115 which is not used by the BDOS. If a BDOS call is made with 115 in the C register, then it is trapped and sent to the GDOS. The GDOS is passed a block of pointers that is analogous to the FCB for files. This block of pointers is called a parameter list and a pointer to this list is passed in the DE register pair. (ie: the address of the parameter block is passed in DE). This parameter block contains all the information that is passed in either direction.

Using assembler as an example:-

;to open the workstation, the following routine might be used

```
GDOS:  mvi    c,115    ;BDOS function for all GDOS calls
        lxi    d,Parms
        jmp    Bdos    ;call location 0005H

parms:  ;our parameter block
        dw     ctrl    ;address of control array
        dw     intin   ;address of input parameter array. This is the array
                        ;of integer information sent to the GDOS
        dw     ptsin   ;address of input point coordinate array. This is the
                        ;array of coordinates sent to the GDOS
        dw     intout  ;address of output parameter array. This is the array
                        ;of integer information sent from the GDOS
        dw     ptsout  ;address of output point coordinate array. This is the
                        ;array of coordinate information sent from the GDOS.
```

;the control array can contain a number of entries  
;but the first five entries are invariant in meaning

```
ctrl:
opcode: dw     01      ;this is set to the desired opcode
Iverts: dw     00      ;this must be set to the number of vertices
                        ;(coordinate pairs) sent in PISIN array
Overts: dw     00      ;set by GDOS to the length of the output
                        ;parameter array in PISOUT
Iparms: dw     10      ;the length of the input parameter array INTIN
Outputs: dw     00      ;set by GDOS to the number of vertices in the
                        ;output point array INTOUT
```

we can represent this in the following C function:-

```
/*-----*/
GDOS() /* call the GDOS */
{
    bdos(115,&the_parameters);
}
```

The parameter block consists of an array of 16-bit pointers to various arrays. We need to set up this array before we call the GDOS

```
struct parameter_block
{
    int    *ad_ctrl;    /* address of control array */
    int    *ad_intin;   /* address of input parameter array */
    int    *ad_ptsin;   /* address of input point coordinate array */
    int    *ad_intout;  /* address of output parameter array */
    int    *ad_ptsout;  /* address of output point coordinate array */
} the_parameters;
```

The most important of the arrays is the control array. This is an integer array containing such vital bits of information such as the actual GSX opcode desired, and the length of any coordinate arrays.

```
struct a_control_structure
{
    int    opcode;      /* input and output parameters */
    int    ptsin_vertices; /* number of vertices in array ptsin. Each
                           consists of an X and Y coordinate pair as integers */
    int    ptsout_vertices; /* number of vertices in array ptsout. Each
                             consists of an X and Y coordinate pair as integers */
    int    intin_length; /* length of intin, Array of integer input
                           parameters */
    int    intout_length; /* length of intout, output point parameters */
    int    ctrl_6;        /* These have uses depending on opcode */
    int    ctrl_7;
    int    ctrl_8;
    int    ctrl_9;
    int    ctrl_10;
} control;
```

The idea of a vertex array may seem a bit foreign. In fact, it is fairly simple. A cartesian point is defined by its XY coordinate. All coordinates in GSX are Normalized Device coordinates. To obtain device-independence, these are scaled between 0 to 32767 in both axes, rather than for the physical device. In other words, GSX understands a normalized space of 32K by 32K; When a location is described, it is sent as a pair of integer coordinates (vertices). Normally, one wants to specify lines, shapes and areas so one wants to send a whole series of points. One therefore presents a series of vertex pairs to GSX. This is in the form of an integer array. If you are making a call to GSX that requires an array of vertex pairs, one quite simply passes a pointer to it in the parameter block.

For the time being, lets not be so ambitious. Lets just open the workstation. To do this, one should refer to GDOS Opcode 1 in section 9.5. It can be seen from this that we need to specify the workstation type and the defaults for the various colours, styles and sizes. We do this by passing a pointer to a structure as follows:-

```
struct our_defaults
{
    int device;          /* which device driver to load */
    int linestyle;      /* the type of line to draw */
    int polyline_colour_index; /* the colour of line to draw */
    int marker_type;    /* the type of marker to draw */
}
```

```

    int polymarker_colour_index; /* the colour of marker to draw */
    int text_font;               /* the text font to use */
    int text_colour_index;       /* the text colour */
    int fill_interior_style;      /* the interior fill style */
    int fill_style_index;         /* the type of hatching/halftone */
    int fill_colour_index;        /* the colour of fill */
}
defaults = {
    0, /* device driver id, workstation identifier */
/* 1-10=CRT, 11-20=plotter, 21-30=Printer, 31-40=Metafile, 41-50 other */
    1, /* the line type */
/* 1=solid, 2=dashed, 3=dotted, 4=dash,dot, 5=long dash */
    1, /* the polyline colour index */
/* 0=device maximum */
    1, /* the marker type */
/* 1=dot, 2=plus, 3=asterix, 4=circle, and 5=diagonal cross */
    1, /* the polymarker colour index */
/* 0=device maximum */
    1, /* the text font */
/* 1=device maximum */
    1, /* the text colour index */
/* 0=device maximum */
    1, /* the fill interior style */
/* 0=hollow, 1=solid, 2=halftone, 3=hatch */
    1, /* fill style index */
/* 1=vertical line, 2=horizontal line, 3=+45°, 4=-45°, 5=cross, 6=X */
    1 /* fill colour index */
/* 0=device maximum */
};

```

When a workstation is opened, it fills in an integer array that provides both GSX and the calling program with interesting information about the device. This array is structured as follows:-

```

struct our_device
{
    int width_in_rasters;
    int height_in_rasters;
    int cannot_scale;
    int pixel_width;
    int pixel_length;
    int no_character_sizes;
    int no_linetypes;
    int no_linewidths;
    int no_types_markers;
    int no_sizes_markers;
    int no_fonts;
    int no_patterns;
    int no_hatch_styles;
    int no_colours;
    int no_GDPs;
    int list_of_GDPs[10];
    int GDP_attributes[10];
    int has_colour;
    int can_rotate;
    int can_fill;
    int can_read_cell;

```



```

int colours;
int no_locators;
int no_valuators;
int no_choice_devices;
int no_string_devices;
int workstation_type;
) specification;

```

This array has to be created, and its location passed to GSX. When the workstation is opened, you will find that it has been filled with the information that is relevant for that workstation. There is a second array with more information about the way lines, characters and markers are displayed.

```

struct our_dimensions
(
    int spare1;
    int min_char_height;
    int spare2;
    int max_char_height;
    int min_line_width;
    int spare3;
    int max_line_width;
    int spare4;
    int spare5;
    int min_marker_height;
    int spare6;
    int max_marker_height;
) dimensions;

```

With all these structures defined, we are ready to open the workstation. Here is the function that does it.

```

/*-----*/
open_workstation(device) /* opens the workstation, sets the defaults,
and provides the calling program with information about the device */
int device;

{
    defaults.device=device;
    the_parameters.ad_control=&control; /* lets keep this so! */
    the_parameters.ad_intin=&defaults; /* tell the BIOS of our defaults */
    the_parameters.ad_intout=&specification; /* get the device specification */
    the_parameters.ad_ptsout=&dimensions; /* and dimensions of device stuff */

    control.opcode=1;
    control.ptsin_vertices=0; /* number of vertices in array ptsin. Each
                                consists of an X and Y coordinate pair as integers */
    control.intin_length=10; /* length of intin, Array of integer input
                                parameters */
    BIOS();
}

```

Flushed with success, we can now write the 'close\_workstation' function. This is a great deal easier.

```

/*-----*/

```

```

close_workstation()    /* closes the workstation. If it is a CRT, it blanks
the screen and selects alpha mode, if it is a printer or plotter then it
updates the page and shuts off. */
{
control.opcode=2;
control.ptsin_vertices=0;
GDOS();
}

```

Note that it is important that all calls to GSX must initialise the 'ptsin\_vertices' variable. This tells GSX how many coordinate pairs are being passed to GSX. It should be set to zero if no vertices are passed.

We are now ready for our first GSX program. This is not really very much but it is really quite useful for finding out about device drivers. It displays most of the information passed back on the 'open\_workstation' command. We include a header file which defines all the structures that we use and declares the actual arrays as being external. We link in the 'open\_workstation' and 'close\_workstation' commands. In this way we keep things relatively neat and orderly. It is too easy to allow a GSX program to become too complex.

```

#include <D:GSX.H>

main()
{
open_workstation(0);    /* to get all the driver dimensions */
close_workstation();    /* as we need to write in alpha mode */

printf("The following are the device parameters:-\n\n");

printf("The device width in rasters=%d, the device height in rasters = %d\n",
specification.width_in_rasters, specification.height_in_rasters);
printf("Images ");
if (specification.cannot_scale) printf("cannot"); else printf ("can");
printf(" be scaled \n");
printf("Width of pixel=%d height of pixel = %d\n",
specification.pixel_width, specification.pixel_length);
printf("There are %d linetypes, %d linewidths, and %d marker types\n",
specification.no_linetypes, specification.no_linewidths,
specification.no_types_markers);
printf("There are %d marker sizes, %d character sizes and %d fonts\n",
specification.no_sizes_markers, specification.no_character_sizes,
specification.no_fonts);
printf("There are %d patterns, %d hatch styles and %d colours\n",
specification.no_patterns, specification.no_hatch_styles,
specification.no_colours);
printf("There are %d GDSs. It ",specification.no_GDSs);
if (specification.has_colour) printf("has"); else printf ("has'nt");printf(" colour, It ");
if (specification.can_rotate) printf("can"); else printf ("cannot");printf(" rotate characters,\n");
printf ("Polylines ");
if (specification.can_fill) printf("can"); else printf ("cannot");printf(" be filled, One ");
if (specification.can_read_cell)
printf("can"); else printf ("cannot");printf(" read cell arrays \n");
printf("There are %d colours. ",specification.colours);
printf("there are %d locators, %d valutors and %d choice devices\n",
specification.no_locators,specification.no_valutors,specification.no_choice_devices);

```

```

printf("there are %d string devices. The workstation type is %d\n\n",
specification.no_string_devices,specification.workstation_type);
printf("the minimum character height is %d, the maximum character height is %d\n",
dimensions.min_char_height, dimensions.max_char_height);
printf("the minimum line width is %d, the maximum line width is %d\n",
dimensions.min_line_width, dimensions.max_line_width);
printf("the minimum marker height is %d, the maximum marker height is %d\n",
dimensions.min_marker_height, dimensions.max_marker_height);
)

```

This program simply accesses the various components in the specifications structure and the auxilliary dimensions structure. Normally, one is rarely required to access this information, but it is there, and is accessible.

Having opened a workstation, the time has come for some simple line drawing. To do this, we need to define a function to do this. It takes two parameters; the address of the polyline vertex array and the number of vertices in the array.

```

/*-----*/
polyline(array, vertices) /* draws the polyline array addressed by array,
containing 'vertices' vertices */
{
control.opcode=6;
control.ptsin_vertices=vertices;
the_parameters.ad_ptsin=array; /* address of input point coordinate array */
GDOS();
}

```

This is a very powerful function and is the basis of most graphics. Most applications draw all its characters out of polylines, and it is, in fact, possible to produce acceptable graphics using no other GDOS call. All one needs to do is to create the integer array. EG:

```

square[10]= /* X      Y */
(
    400,    400,
    15000,  400,
    15000,  15000,
    400,    15000,
    400,    400);

polyline(square,5);

```

With the GSX calls we have so far, we can do something more useful. We will need, at some time, the facility to draw graphic characters on the screen so let us go ahead and construct a character font. We can then use this to draw characters in various sizes and orientations on the screen. Character fonts can take up a great deal of space unless one can compress the data a bit so we will choose to compress each XY vertex into one byte. This enables us to map the character on a private coordinate space of 16\*16. We choose to have the origin in the centre of this space, with coordinates going from -7 to +7. This simplifies rotation of the shape. For characters, rather than other graphic shapes, we keep to the positive quadrant 0 to +8 rather than use the entire space. Due to the fact that, historically speaking, our character set was

designed to be drawn easily with a chisel, we can do perfectly good characters within what might seem a rather unpromising cramped grid. A stroke font is rather useful to have so we will show it in its entirety. It will be compiled and stored away in our relocatable library file.

# /\* STROKE FONT CHARACTER SET

The following byte arrays are compressed vertice information for a stroked character set. The X,Y coordinate information is represented by 4 bit 2's complement numbers in the range of +7 X, +7 Y. End of character is represented by -8 X, -8 Y and a dark vector (lifting the pen) is represented by -8 X, 0 Y.

```

      BIT   7 6 5 4 3 2 1 0
            \ / \ /
             X  Y

```

ASCII characters are currently mapped into the positive quadrant, with the origin at the lower left corner of an upper case character. This helps with character rotation \*/

```

#define      END_CHARACTER      0x88      /* end of character */
#define      DARK_VECTOR      0x80      /* last vector of polyline */

```

```

char font_20[]={END_CHARACTER};
char font_21[]={0x20,0x21,DARK_VECTOR,0x23,0x26,END_CHARACTER};
char font_22[]={0x24,0x26,DARK_VECTOR,0x54,0x56,END_CHARACTER};
char font_23[]={0x20,0x26,DARK_VECTOR,0x40,0x46,DARK_VECTOR,0x04,0x64,
                DARK_VECTOR,0x02,0x62,END_CHARACTER};
char font_24[]={0x2F,0x27,DARK_VECTOR,0x01,0x10,0x30,0x41,0x42,0x33,0x13,
                0x04,0x05,0x16,0x36,0x045,END_CHARACTER};
char font_25[]={0x11,0x55,DARK_VECTOR,0x14,0x15,0x25,0x24,0x14,DARK_VECTOR,
                0x41,0x51,0x52,0x42,0x41,END_CHARACTER};
char font_26[]={0x50,0x14,0x15,0x26,0x36,0x45,0x44,0x10,0x30,0x52,
                END_CHARACTER};
char font_27[]={0x34,0x36,END_CHARACTER};
char font_28[]={0x4E,0x11,0x14,0x47,END_CHARACTER};
char font_29[]={0x0E,0x31,0x34,0x07,END_CHARACTER};
char font_2A[]={0x30,0x36,DARK_VECTOR,0x11,0x55,DARK_VECTOR,0x15,0x51,
                DARK_VECTOR,0x03,0x63,END_CHARACTER};
char font_2B[]={0x30,0x36,DARK_VECTOR,0x03,0x63,END_CHARACTER};
char font_2C[]={0x11,0x20,0x2F,0x0D,END_CHARACTER};
char font_2D[]={0x03,0x63,END_CHARACTER};
char font_2E[]={0x00,0x01,0x11,0x10,0x00,END_CHARACTER};
char font_2F[]={0x00,0x01,0x45,0x46,END_CHARACTER};
char font_30[]={0x01,0x05,0x16,0x36,0x45,0x41,0x30,0x10,0x01,END_CHARACTER};
char font_31[]={0x04,0x26,0x20,DARK_VECTOR,0x00,0x040,END_CHARACTER};
char font_32[]={0x05,0x16,0x36,0x45,0x44,0x00,0x40,0x041,END_CHARACTER};
char font_33[]={0x05,0x16,0x36,0x45,0x44,0x33,0x42,0x41,0x30,0x10,0x01,
                DARK_VECTOR,0x13,0x033,END_CHARACTER};
char font_34[]={0x06,0x03,0x043,DARK_VECTOR,0x20,0x026,END_CHARACTER};
char font_35[]={0x01,0x10,0x30,0x41,0x42,0x33,0x03,0x06,0x046,END_CHARACTER};

```

```

char font_36[]=(0x02,0xd3,0x33,0x42,0x41,0x30,0x10,0x01,0x05,0xd6,0x36,
               0x045,END_CHARACTER);
char font_37[]=(0x06,0x46,0x44,0x00,END_CHARACTER);
char font_38[]=(0x01,0x02,0xd3,0x04,0x05,0xd6,0x36,0x45,0x44,0x33,0x42,
               0x41,0x30,0x10,0x01,DARK_VECTOR,0xd3,0x023,END_CHARACTER);
char font_39[]=(0x01,0x10,0x30,0x41,0x45,0x36,0xd6,0x05,0x04,0xd3,0x33,0x044,
               END_CHARACTER);
char font_3A[]=(0x15,0x25,0x24,0xd4,0xd5,DARK_VECTOR,0xd2,0x22,0x21,0xd1,
               0xd2,END_CHARACTER);
char font_3B[]=(0x15,0x25,0x24,0xd4,0xd5,DARK_VECTOR,0x21,0xd1,0xd2,0x22,
               0x20,0xdF,END_CHARACTER);
char font_3C[]=(0x30,0x03,0x036,END_CHARACTER);
char font_3D[]=(0x02,0x042,DARK_VECTOR,0x04,0x044,END_CHARACTER);
char font_3E[]=(0x10,0x43,0x16,END_CHARACTER);
char font_3F[]=(0x06,0xd7,0x37,0x46,0x45,0x34,0x24,0x022,DARK_VECTOR,0x21,
               0x020,END_CHARACTER);
char font_40[]=(0x50,0x10,0x01,0x06,0xd7,0x57,0x66,0x63,0x52,0x32,0x23,
               0x24,0x35,0x55,0x064,END_CHARACTER);
char font_41[]=(0x00,0x04,0x26,0x44,0x040,DARK_VECTOR,0x03,0x043,
               END_CHARACTER);
char font_42[]=(0x00,0x06,0x36,0x45,0x44,0x33,0x42,0x41,0x30,0x00,
               DARK_VECTOR,0x03,0x033,END_CHARACTER);
char font_43[]=(0x45,0x36,0x16,0x05,0x01,0x10,0x30,0x041,END_CHARACTER);
char font_44[]=(0x00,0x06,0x36,0x45,0x41,0x30,0x00,END_CHARACTER);
char font_45[]=(0x40,0x00,0x06,0x046,DARK_VECTOR,0x03,0x023,END_CHARACTER);
char font_46[]=(0x00,0x06,0x046,DARK_VECTOR,0x03,0x023,END_CHARACTER);
char font_47[]=(0x45,0x36,0x16,0x05,0x01,0x10,0x30,0x41,0x43,0x023,
               END_CHARACTER);
char font_48[]=(0x00,0x06,DARK_VECTOR,0x03,0x043,DARK_VECTOR,0x40,0x046,
               END_CHARACTER);
char font_49[]=(0x10,0x30,DARK_VECTOR,0x20,0x026,DARK_VECTOR,0xd6,0x036,
               END_CHARACTER);
char font_4A[]=(0x01,0x10,0x30,0x41,0x046,END_CHARACTER);
char font_4B[]=(0x00,0x06,DARK_VECTOR,0x02,0x046,DARK_VECTOR,0xd3,0x040,
               END_CHARACTER);
char font_4C[]=(0x40,0x00,0x06,END_CHARACTER);
char font_4D[]=(0x00,0x06,0x24,0x46,0x040,END_CHARACTER);
char font_4E[]=(0x00,0x06,DARK_VECTOR,0x05,0x041,DARK_VECTOR,0x40,0x046,
               END_CHARACTER);
char font_4F[]=(0x01,0x05,0xd6,0x36,0x45,0x41,0x30,0xd0,0x01,END_CHARACTER);
char font_50[]=(0x00,0x06,0x36,0x45,0x44,0x33,0x03,END_CHARACTER);
char font_51[]=(0xd2,0x30,0x10,0x01,0x05,0xd6,0x36,0x45,0x41,0x30,
               END_CHARACTER);
char font_52[]=(0x00,0x06,0x36,0x45,0x44,0x33,0x03,DARK_VECTOR,0xd3,0x040,
               END_CHARACTER);
char font_53[]=(0x01,0x10,0x30,0x41,0x42,0x33,0xd3,0x04,0x05,0xd6,0x36,
               0x045,END_CHARACTER);
char font_54[]=(0x06,0x046,DARK_VECTOR,0x20,0x026,END_CHARACTER);
char font_55[]=(0x06,0x01,0x10,0x30,0x41,0x046,END_CHARACTER);
char font_56[]=(0x06,0x02,0x20,0x42,0x046,END_CHARACTER);
char font_57[]=(0x06,0x00,0x22,0x40,0x046,END_CHARACTER);
char font_58[]=(0x00,0x01,0x45,0x046,DARK_VECTOR,0x40,0x41,0x05,0x06,
               END_CHARACTER);
char font_59[]=(0x06,0x24,0x020,DARK_VECTOR,0x24,0x46,END_CHARACTER);
char font_5A[]=(0x06,0x46,0x45,0x01,0x00,0x40,END_CHARACTER);
char font_5B[]=(0x37,0xd7,0xdF,0x3F,END_CHARACTER);
char font_5C[]=(0x06,0x05,0x41,0x40,END_CHARACTER);
char font_5D[]=(0xd7,0x37,0x3F,0x2F,END_CHARACTER);

```

```

char font_5E[]={0x04,0x26,0x044,END_CHARACTER};
char font_5F[]={0x0F,0x07F,END_CHARACTER};
char font_60[]={0x54,0x36,END_CHARACTER};
char font_61[]={0x40,0x43,0x34,0x14,0x03,0x01,0x10,0x30,0x041,END_CHARACTER};
char font_62[]={0x06,0x01,0x10,0x30,0x41,0x43,0x34,0x14,0x03,END_CHARACTER};
char font_63[]={0x41,0x30,0x10,0x01,0x03,0x14,0x34,0x043,END_CHARACTER};
char font_64[]={0x46,0x41,0x30,0x10,0x01,0x03,0x14,0x34,0x43,END_CHARACTER};
char font_65[]={0x41,0x30,0x10,0x01,0x03,0x14,0x34,0x43,0x42,0x02,
END_CHARACTER};
char font_66[]={0x20,0x25,0x36,0x46,0x55,DARK_VECTOR,0x03,0x43,END_CHARACTER};
char font_67[]={0x41,0x30,0x10,0x01,0x03,0x14,0x34,0x43,0x4F,0x3E,0x1E,0x0F,
END_CHARACTER};
char font_68[]={0x00,0x06,DARK_VECTOR,0x03,0x14,0x34,0x43,0x40,END_CHARACTER};
char font_69[]={0x20,0x23,DARK_VECTOR,0x25,0x26,END_CHARACTER};
char font_6A[]={0x46,0x45,DARK_VECTOR,0x43,0x4F,0x3E,0x1E,0x0F,END_CHARACTER};
char font_6B[]={0x00,0x06,DARK_VECTOR,0x01,0x34,DARK_VECTOR,0x12,0x30,
END_CHARACTER};
char font_6C[]={0x20,0x26,END_CHARACTER};
char font_6D[]={0x00,0x04,DARK_VECTOR,0x03,0x14,0x23,0x34,0x43,0x40,
END_CHARACTER};
char font_6E[]={0x00,0x04,DARK_VECTOR,0x03,0x14,0x34,0x43,0x40,END_CHARACTER};
char font_6F[]={0x01,0x03,0x14,0x34,0x43,0x41,0x30,0x10,0x01,END_CHARACTER};
char font_70[]={0x04,0x0E,DARK_VECTOR,0x01,0x10,0x30,0x41,0x43,0x34,0x14,
0x03,END_CHARACTER};
char font_71[]={0x41,0x30,0x10,0x01,0x03,0x14,0x34,0x43,DARK_VECTOR,0x44,
0x4E,END_CHARACTER};
char font_72[]={0x00,0x04,DARK_VECTOR,0x03,0x14,0x34,END_CHARACTER};
char font_73[]={0x01,0x10,0x30,0x41,0x32,0x12,0x03,0x14,0x34,0x43,
END_CHARACTER};
char font_74[]={0x04,0x44,DARK_VECTOR,0x26,0x21,0x30,0x40,0x51,END_CHARACTER};
char font_75[]={0x04,0x01,0x10,0x30,0x41,DARK_VECTOR,0x44,0x40,END_CHARACTER};
char font_76[]={0x04,0x02,0x20,0x42,0x44,END_CHARACTER};
char font_77[]={0x04,0x00,0x22,0x40,0x44,END_CHARACTER};
char font_78[]={0x00,0x44,DARK_VECTOR,0x04,0x40,END_CHARACTER};
char font_79[]={0x04,0x01,0x10,0x30,0x41,DARK_VECTOR,0x44,0x4F,0x3E,0x1E,
0x0F,END_CHARACTER};
char font_7A[]={0x04,0x44,0x00,0x40,END_CHARACTER};
char font_7B[]={0x40,0x11,0x32,0x03,0x34,0x15,0x46,END_CHARACTER};
char font_7C[]={0x20,0x23,DARK_VECTOR,0x25,0x27,END_CHARACTER};
char font_7D[]={0x00,0x31,0x12,0x43,0x14,0x35,0x06,END_CHARACTER};
char font_7E[]={0x06,0x27,0x46,0x67,END_CHARACTER};
char font_7F[]={0x07,0x77,END_CHARACTER};

```

/\* this is a table of pointers to the actual compressed character arrays \*/  
/\* in this table, all control codes are represented by a space \*/

```

char *font_table[]={
font_20,font_20,font_20,font_20,font_20,font_20,font_20,font_20,
font_20,font_20,font_20,font_20,font_20,font_20,font_20,font_20,
font_20,font_20,font_20,font_20,font_20,font_20,font_20,font_20,
font_20,font_20,font_20,font_20,font_20,font_20,font_20,font_20,
font_20,font_21,font_22,font_23,font_24,font_25,font_26,font_27,
font_28,font_29,font_2A,font_2B,font_2C,font_2D,font_2E,font_2F,
font_30,font_31,font_32,font_33,font_34,font_35,font_36,font_37,
font_38,font_39,font_3A,font_3B,font_3C,font_3D,font_3E,font_3F,
font_40,font_41,font_42,font_43,font_44,font_45,font_46,font_47,
font_48,font_49,font_4A,font_4B,font_4C,font_4D,font_4E,font_4F,

```

```
font_50,font_51,font_52,font_53,font_54,font_55,font_56,font_57,
font_58,font_59,font_5A,font_5B,font_5C,font_5D,font_5E,font_5F,
font_60,font_61,font_62,font_63,font_64,font_65,font_66,font_67,
font_68,font_69,font_6A,font_6B,font_6C,font_6D,font_6E,font_6F,
font_70,font_71,font_72,font_73,font_74,font_75,font_76,font_77,
font_78,font_79,font_7A,font_7B,font_7C,font_7D,font_7E,font_7F
);
```

Now we have our stroke font. How do we use it? Here is a simple example, where a file is written, character by character, to the device so as to display an entire page on the workstation. This is really only practical on a plotter as the individual characters are so small. (it actually crashes most of the dot-matrix drivers). On a plotter, it will write out the file, character by character and page by page.

```
#include <D:GSX.H>
#include <B:STDIO.H>

int scratch[30];
int x;
int y;
int size=32;          /* our scaling factor */
int margin=22;         /* 22 character margin width */
int linefeed=510;      /* NDCs to decrement for a linefeed */
int x_cursor;
int y_cursor;
char str_buffer[82];    /* The buffer for the each line */

main()
{
    int line=0;
    FILE *fp;

    y_cursor=0x07FFF-(5*512);
    if (argc != 2){ printf("please give the name of the file to print"); exit(1);}
    fp= fopen (argv[1],"r");
    if (fp=NULL)
    {
        printf ("cannot find the file %s",argv[1]); exit(1);
    }
    open_workstation(11); /* lets use the plotter */
    while (fgets(str_buffer,80,fp))
    {
        /* get each line in turn */
        x_cursor=22*(8*32); /* bump over the margin */
        do_text(str_buffer); /* print out the line */
        if (line++ > 52) /* do fiftytwo lines per page */
        {
            clear_workstation(); /* do a formfeed */
            line=0; /* zap the linecount */
            y_cursor=0x07FFF-(5*512);
        }
        y_cursor -= linefeed; /* move our cursor down one line */
    }
    close_workstation();
    fclose(fp);
}
```

```

/*-----*/
do_text(string) /* send out the text starting at the location
X_cursor,Y_cursor with the west/east orientation and current size */
{
    register char character;
    while (character=*string++)
    {
        display(character); /* display the character */
        x_cursor += (size*8); /* and bump the cursor */
    }
}

/*-----*/
display(character) /* display the character at the 'cursor' position */
char character;
{
    char *where;
    int count=0;
    char byte;

    where=font_table[character]; /* point to the table entry */

    while (-1)
    {
        if (((byte=*where++) == END_CHARACTER) || (byte == DARK_VECTOR))
        {
            if (count) polyline(scratch, count>>1); /* draw the character */
            count = 0;
            if (byte == END_CHARACTER) return();
        }
        else
        {
            /* expand the stroke data into XY coordinates */
            y = byte & 0x0F; /* get Y coordinate */
            if (byte & 0x08) y = byte | 0xFF0; /* sign extend character */
            x = byte & 0xF0 >> 4; /* get X coordinate */
            if (byte & 0x08) x = byte | 0xFF0; /* sign extend character */
            x=x*size; /* scale up the coordinates */
            y=y*size;
            scratch[count++]=x+x_cursor; /* put them in the coordinate */
            scratch[count++]=y+y_cursor; /* space */
        }
    }
}

```

So here we have a GSX program that has required nothing more than the workstation commands and the polyline command. Naturally, one could expand the ideas of this little program into a personal typesetting system, with the facility to type text in any size or direction. One would need to use a plotter or laser printer to get the required accuracy, and you will outgrow the supplied font in favour of a more complex proportional font. Naturally, if the program is dedicated to this task, one has 40K of TPA (in CP/M+ to play with so the font need not be so compressed. There already exist books of 'stroke' fonts



but it might be more interesting to design your own.

It may seem rather perverse to demonstrate a graphics system with a program to draw characters, but once this is achieved, the door is open to much more sophisticated operations. The idea of representing an image as a polyline is central to GSX. Images are drawn, filled, and outlined with markers, all with identical data structures. A polyline shape in a private space can be mapped into the GSX coordinate system in any size or rotation. In this demonstration of using GSX we have shown you how to get started, and how to get to grips with the polyline. To expand and develop these ideas into a useful program is relatively easy once the GDOS calls and their calling conventions are known. Now that we can make a GDOS call, The next task is to outline the GDOS calls.

## 9.5 The GDOS Functions.

The GDOS functions are as follows:

opcode 1	-initialize the graphics screen
opcode 2	-Stop graphics output to the screen
opcode 3	-Clear the screen
opcode 4	-Display all pending graphics
opcode 5	-Escape (see above) device-dependent ops
opcode 6	-Output a polyline
opcode 7	-Output markers
opcode 8	-Output graphic text
opcode 9	-Display and fill a polygon
opcode 10	-Define a cell array
opcode 11	-Display a drawing primitive
opcode 12	-Set the text size
opcode 13	-Set text direction
opcode 14	-define colour representation
opcode 15	-Set the linestyle for lines
opcode 17	-Set colour for lines
opcode 18	-Set the marker type
opcode 19	-Set the marker size
opcode 20	-Set colour for markers
opcode 21	-Set the text font
opcode 22	-Set the text colour
opcode 23	-Set interior style of fill
opcode 24	-Set Fill Style
opcode 25	-Set colour for polygon fill
opcode 26	-return colour representation.
opcode 27	-Inquire the cell array
opcode 28	-return value of locator
opcode 29	-return value of valuator
opcode 30	-return value of choice device
opcode 31	-Return character string
opcode 32	-Set current writing mode
opcode 33	-Set the input mode

opcode 5 gives access to console-oriented facilities

- 1/ Inquire addressable character cells
- 2/ Exit Graphics mode
- 3/ Enter Graphics Mode
- 4/ Cursor Up
- 5/ Cursor Down
- 6/ Cursor Right
- 7/ Cursor Left
- 8/ Home Cursor
- 9/ Erase to end of screen
- 10/ Erase to end of line
- 11/ Direct cursor address
- 12/ Output Cursor Addressable Text
- 13/ Reverse Video on
- 14/ Reverse Video off
- 15/ Inquire current cursor address
- 16/ Inquire the tablet status
- 17/ Hardcopy
- 18/ Place cursor at location
- 19/ Remove cursor

For all these functions, some of the parameters passed to GDOS within the five arrays must be correctly set. In some cases, these are minimal, but where, for example, a complex figure is to be drawn or filled, or a workstation is to be opened, much information has to be passed between GDOS and the calling program.

The GSX functions can be grouped according to their purpose

#### A) GSX Control functions

These commands deal with the fundamental business of opening, updating, clearing and closing a particular driver (or workstation).

opcode 1	-initialize the workstation
opcode 2	-Stop graphics output to the Workstation
opcode 3	-Clear the workstation
opcode 4	-Display all pending graphics

#### B) GSX Mode functions

These are used to change the default settings for the colours and the modes with which lines, area fill and graphic text is written.

opcode 14	-define colour representation
opcode 26	-return colour representation.
opcode 32	-Set current writing mode

#### C) GSX Line Drawing Functions

These commands deal with the mechanism by which lines are drawn or erased and the style in which they are drawn.

opcode 6	-Output a polyline
opcode 11	-generalized drawing primitive
opcode 15	-Set the linestyle for lines
opcode 16	-Set polyline width
opcode 17	-Set colour for lines

#### D) GSX Marker Drawing Functions (works only partially in the DDSCREEN.PRL)

These commands deal with the drawing and erasure of markers. Markers are used in business and scientific graphs, particularly in line and scattergraphs. They are useful, as well for other applications where dots, crosses and other location indicators are required.

opcode 7	-Output a polymarker
opcode 18	-Set the marker type
opcode 19	-Set the marker size
opcode 20	-Set colour for markers

#### E) GSX Filled Area Functions (this will not work in the DDSCREEN.PRL)

These commands deal with area fill. Areas are defined in exactly the same way as polygons, by a polyline. The area is defined by its periphery, and it is assumed that the first and last defined point (or vertex) are joined to form a closed shape.

opcode 9	-polygon fill
opcode 23	-Set interior style of fill
opcode 24	-Set Fill Style
opcode 25	-Set colour for polygon fill

F) GSX Graphic Text Functions (these work only partially in the DDSCREEN.PRL)

Graphic text is distinct from ordinary text in that it coexists with other graphics such as polylines, markers and shapes, and can appear in all the available colours. It can, in a full implementation, be rotated or sized precisely.

opcode 8	-Output graphic text
opcode 12	-Set the text size
opcode 13	-Set text direction
opcode 21	-Set the text font
opcode 22	-Set the text colour

G) GSX Graphic cursor and GSX Alpha Text Functions

This call gives access to all the normal terminal features of a workstation and generally deals with screens of text. Menus, forms and other text screens are done using this call. There are also subcodes to deal with graphic cursor movement and hardcopy.

opcode 5	-Escape (see subcodes above)
----------	------------------------------

H) GSX Cell Array functions (these will not work in the DDSCREEN.PRL)

Cell arrays were an attempt to deal with bitmap-oriented rather than vector-oriented graphics. The system works, and is useful when involved with complex colour graphics, but it tends to be rather slow and is not always implemented on graphics consoles.

opcode 10	-Fill Cell Array
opcode 27	-Inquire cell array

I) GSX Graphic input functions (these will not work in the DDSCREEN.PRL)

These are essential for a graphics workstation. They deal with inputting graphics and keyboard information. Locations on a graphics screen can be selected using a device such as a mouse, trackball, or cursor keys.

opcode 28	-return value of locator
opcode 29	-return value of valuator
opcode 30	-return value of choice device
opcode 31	-Return character string
opcode 33	-Set the input mode

-----+-----  
GDOS OPCODE 1 - Open Workstation

This loads the workstation graphic driver and sets it to become the current device. It is initialised to the settings in the input array and information is returned to the calling program. The device is blanked.

## Input parameters

## Control array

```

Contrl(1) -- Opcode = 1
Contrl(2) -- Number of vertices= 0
Contrl(4) -- Length of INTIN = 10
Contrl(6) -- 0
Contrl(7) -- 0
Contrl(8) -- 0
Contrl(9) -- 0

```

## Settings of the configurable options for the device

```

Intin(1) -- Workstation identifier
Intin(2) -- Linetype
Intin(3) -- Polyline colour index
Intin(4) -- Marker Type
Intin(5) -- Polymarker colour index
Intin(6) -- Text font
Intin(7) -- Text colour index
Intin(8) -- Fill interior style
Intin(9) -- Fill style index
Intin(10) -- Fill colour index

```

## Output parameters

```

Contrl(3) -- Number of Output vertices= 6
Contrl(5) -- Length of INTOUT = 45
Intout(1) -- Width of device in rasters(dots)
Intout(2) -- Height of device in rasters(dots)
Intout(3) -- Are images imprecisely scaled?
                (Yes=1, No=0)
Intout(4) -- Width of a pixel in micrometers
Intout(5) -- Height of a pixel in micrometers
Intout(6) -- No. of character heights(0=continuous)
Intout(7) -- Number of linetypes
Intout(8) -- Number of Line widths
Intout(9) -- Number of marker types
Intout(10) -- Number of marker sizes
Intout(11) -- Number of fonts
Intout(12) -- Number of patterns
Intout(13) -- Number of hatch styles
Intout(14) -- Number of predefined colours
Intout(15) -- No. of Generalised Drawing primitives
Intout(16-25) -- List of GDPs supported. (1=bar. 2=arc,
                3=pie slice, 4=circle, 5=ruling chars, -1=end of list)
Intout(26-35) -- List of GDP's attribute set(1=polyline,
                1=polymarker, 2=text, 3=fill area, 4=none. -1=end)
Intout(36) -- Is there colour capability?(Yes=1 No=0)
Intout(37) -- Can text be rotated? (Yes=1 No=0)
Intout(38) -- Can areas be filled? (Yes=1 No=0)

```

```

|   Intout(39) --   Can cell arrays be read? (Yes=1 No=0) |
|   Intout(40) --   Number of available colours.         |
|   Intout(41) --   Number of available devices.         |
|   Intout(42) --   Number of available devices.         |
|   Intout(43) --   Number of available devices.         |
|   Intout(44) --   Number of available devices.         |
|   Intout(45) --   Workstation type. (0=output only,    |
|                   1=input only, 2=input/output, 3=Device independent |
|                   segment storage, 4=GKS Metafile.)     |
|   Ptsout(1)  --   0                                     |
|   Ptsout(2)  --   Minimum char height                 |
|   Ptsout(3)  --   0                                     |
|   Ptsout(4)  --   Maximum char. height                |
|   Ptsout(5)  --   Minimum line width                  |
|   Ptsout(6)  --   0                                     |
|   Ptsout(7)  --   Maximum line width                  |
|   Ptsout(8)  --   0                                     |
|   Ptsout(9)  --   0                                     |
|   Ptsout(10) --   Minimum marker height               |
|   Ptsout(11) --   0                                     |
|   Ptsout(12) --   Maximum marker height               |
+-----+
|
+-----+
| GDOS OPCODE  2 - Close Workstation                    |
+-----+
| This operation terminates the device and updates the device |
| (or clears it in the case of a CRT).                     |
|
| Input parameters                                          |
|
|   Control array                                          |
|   Contrl(1)  --   Opcode = 2                            |
|   Contrl(2)  --   Number of vertices= 0                 |
|
| Output parameters.                                       |
|
|   Control array                                          |
|   Contrl(3)  --   No. of output vertices                |
+-----+

```

GDOS OPCODE 3 - Clear Workstation

This operation updates the device and starts a new page (or clears the screen in the case of a CRT).

Input parameters

Control array

Contrl(1) -- Opcode = 3  
Contrl(2) -- Number of vertices= 0

Output parameters.

Control array

Contrl(3) -- No. of output vertices=0

GDOS OPCODE 4 - Update Workstation

This operation updates the device. This has an effect only on such devices as dot-matrix printers that buffer the graphics commands.

Input parameters

Control array

Contrl(1) -- Opcode = 4  
Contrl(2) -- Number of vertices= 0

Output parameters.

Control array

Contrl(3) -- No. of output vertices=0

```

+-----+
| GDOS OPCODE 5 - Perform device-specific operation |
+-----+

```

This operation performs a number of operations related to terminal emulation and the graphics cursor.

Input parameters

Control array

```

|   Contrl(1)  --   Opcode = 4
|   Contrl(2)  --   Number of input vertices
|   Contrl(4)  --   Number of input parameters
|   Contrl(6)  --   Function identifier as follows:-
|       1/ Inquire addressable character cells
|   input: contrl(2)=0, contrl(6)=1, output: contrl(3)=0,
|   intout(1)=rows, intout(2)=columns.
|       2/ Exit Graphics mode
|   input: contrl(2)=0, contrl(6)=2, output: contrl(3)=0,
|       3/ Enter Graphics Mode
|   input: contrl(2)=0, contrl(6)=3, output: contrl(3)=0,
|       4/ Cursor Up
|   input: contrl(2)=0, contrl(6)=4, output: contrl(3)=0,
|       5/ Cursor Down
|   input: contrl(2)=0, contrl(6)=5, output: contrl(3)=0,
|       6/ Cursor Right
|   input: contrl(2)=0, contrl(6)=6, output: contrl(3)=0,
|       7/ Cursor Left
|   input: contrl(2)=0, contrl(6)=7, output: contrl(3)=0,
|       8/ Home Cursor
|   input: contrl(2)=0, contrl(6)=8, output: contrl(3)=0,
|       9/ Erase to end of screen
|   input: contrl(2)=0, contrl(6)=9, output: contrl(3)=0,
|      10/ Erase to end of line
|   input: contrl(2)=0, contrl(6)=10, output: contrl(3)=0,
|      11/ Direct cursor address
|   input: contrl(2)=0, contrl(6)=11, intin(1)=row number,
|   intin(2)=column number. output: contrl(3)=0,
|      12/ Output Cursor Addressable Text
|   input: contrl(2)=0, contrl(4)=Number of characters in string,
|   contrl(6)=12, intin=Text string as ASCII integer array.
|      13/ Reverse Video on
|   input: contrl(2)=0, contrl(6)=13, output: contrl(3)=0,
|      14/ Reverse Video off
|   input: contrl(2)=0, contrl(6)=14, output: contrl(3)=0,
|      15/ Inquire current cursor address
|   input: contrl(2)=0, contrl(6)=15, output: contrl(3)=0,
|   contrl(5)=2, intout(1)=row number, intout(2)=column no.
|      16/ Inquire the tablet status
|   input: contrl(2)=0, contrl(6)=16, output: contrl(3)=0,
|   contrl(5)=1, intout(1)=tablet status
|      17/ Hardcopy
|   input: contrl(2)=0, contrl(6)=17, output: contrl(3)=0,
|      18/ Place graphics cursor at location
|   input: contrl(2)=0, contrl(6)=18, ptsin(1)=X coordinate of
|   location, ptsin(2)=Y coordinate of location. output:
|   contrl(3)=0,
|      19/ Remove cursor
|   input: contrl(2)=0, contrl(6)=19, output: contrl(3)=0,
+-----+

```



```

+-----+
| GDOS OPCODE 6 - Output a polyline |
+-----+
| This operation causes a polyline to be sent to the graphics |
| device. The line starts at the point defined by the first |
| vertex pair in the PTSIN array and is drawn from that point |
| to the next, and from that point to the next and so on. |
+-----+
|
| Input parameters
|
|     Control array
|     Contrl(1) -- Opcode = 4
|     Contrl(2) -- Number of vertices in the polyline
|     Point input array.
|     ptsin(1) -- X coordinate of first point
|     ptsin(2) -- Y coordinate of first point
|     ptsin(3) -- X coordinate of second point
|     ptsin(4) -- Y coordinate of second point
|             etc .... etc
|     ptsin(2n-1)-- X coordinate of point n
|     ptsin(2n) -- Y coordinate of point n
|
| Output parameters.
|
|     Control array
|     Contrl(3) -- No. of output vertices=0
|
+-----+

```

## GDOS OPCODE 7 - Output polymarkers

This operation causes markers to be sent to the graphics device. The first marker is drawn at the point defined by the first vertex pair in the PTSIN array and subsequent markers are drawn at each point in the vertex array. This call enables arrays of coordinate pairs to be depicted on screen for such purposes as business graphics

## Input parameters

## Control array

```

Contrl(1)  -- Opcode = 7
Contrl(2)  -- Number of vertices in the polymarker
Point input array.
ptsin(1)   -- X coordinate of first point
ptsin(2)   -- Y coordinate of first point
ptsin(3)   -- X coordinate of second point
ptsin(4)   -- Y coordinate of second point
etc .... etc
ptsin(2n-1)-- X coordinate of point n
ptsin(2n)  -- Y coordinate of point n

```

## Output parameters.

## Control array

```

Contrl(3)  -- No. of output vertices=0

```

## GDOS OPCODE 8 - Output Graphics Text at XY coordinate

This operation writes text in the current colour, height, rotation and font. Each word of the INTIN array contains only one character.

## Input parameters

## Control array

```

Contrl(1)  -- Opcode = 8
Contrl(2)  -- Number of vertices = 1
Contrl(3)  -- Number of characters in text string
Intin      -- Word character string in integer ASCII
ptsin(1)   -- X coordinate of starting point
ptsin(2)   -- Y coordinate of starting point

```

## Output parameters.

## Control array

```

Contrl(3)  -- No. of output vertices=0

```

```

+-----+
| GDOS OPCODE 9 - Output polyfilled area (polygon) |
+-----+
| This operation causes an area defined by the PTSIN array to |
| be filled in the current fill colour, in the current |
| interior style, and fill style index. |
| |
| Input parameters |
| |
|     Control array |
|     Contrl(1) --   Opcode = 9 |
|     Contrl(2) --   Number of vertices in the polygon |
|     Point input array. |
|     ptsin(1)  --   X coordinate of first point |
|     ptsin(2)  --   Y coordinate of first point |
|     ptsin(3)  --   X coordinate of second point |
|     ptsin(4)  --   Y coordinate of second point |
|     etc .... etc |
|     ptsin(2n-1)-- X coordinate of point n |
|     ptsin(2n)  -- Y coordinate of point n |
| |
| Output parameters. |
| |
|     Control array |
|     Contrl(3) --   No. of output vertices=0 |
+-----+

```

```

+-----+
GDOS OPCODE 10 - Display Cell Array
+-----+

```

This operation causes the device to draw a rectangular array which is defined by the input parameter array XY coordinated and the colour index array. The cell extent is defined by the coordinates of upper right and lower left corners. The colour index array defines the colours for the individual components of the cell.

Input parameters

Control array

```

Contrl(1) -- Opcode = 10
Contrl(2) -- Number of vertices in the cell array
Point input array.
Contrl(4) -- Length of colour index array
Contrl(6) -- Length of each row of array
Contrl(7) -- Number of elements used in each row
Contrl(8) -- Number of rows in colour index
Contrl(9) -- pixel operation to be performed
              1=replace, 2=overstrike, 3=complement and 4=erase
ptsin(1)  -- X coordinate of lower left
ptsin(2)  -- Y coordinate of lower left
ptsin(3)  -- X coordinate of upper right
ptsin(4)  -- Y coordinate of upper right

```

Output parameters.

Control array

```

Contrl(3) -- No. of output vertices=0

```

```

+-----+
| GDOS OPCODE 11 - Output a Generalised Drawing Primitive |
+-----+

```

```

| This operation causes a drawing operation that is built-in |
| to the device to be executed. Some devices, for example,   |
| the capability to draw arcs and circles in one operation   |
| rather than by means of a large polyline operation, and it |
| is advantageous to use this. However, one must ascertain   |
| at 'open_workstation' time what GDPs the device actually has |
| and one must also allow for devices with no GDPs.          |

```

```

| Input parameters

```

```

|     Control array

```

```

|     Contrl(1) -- Opcode = 11
|     Contrl(2) -- No. vertices in the Point input array.
|     Contrl(4) -- Length of input array intin.
|     Contrl(6) -- Primitive I.D.
|     1=BAR with current fill attributes, 2=ARC with current
|     line attributes, 3=PIE SLICE with current fill
|     attributes, 4=CIRCLE with current area fill attributes
|     5=PRINT RULING GRAPHIC CHARACTERS.

```

```

|     Point input array.

```

```

|     ptsin(1) -- X coordinate of first point
|     ptsin(2) -- Y coordinate of first point
|     ptsin(3) -- X coordinate of second point
|     ptsin(4) -- Y coordinate of second point
|     etc .... etc
|     ptsin(2n-1)-- X coordinate of point n
|     ptsin(2n) -- Y coordinate of point n

```

```

|     intin =the data record

```

```

| For the BAR:-

```

```

|     Contrl(2) -- No. vertices =2
|     Contrl(6) -- Primitive I.D.=1
|     ptsin(1) -- X coordinate of lower left of bar
|     ptsin(2) -- Y coordinate of lower left of bar
|     ptsin(3) -- X coordinate of upper right of bar
|     ptsin(4) -- Y coordinate of upper right of bar

```

```

| For the ARC or PIE SLICE:-

```

```

|     Contrl(2) -- No. vertices =4
|     Contrl(6) -- Primitive I.D.=2(ARC) or 3(PIE SLICE)
|     intin(1) -- Start angle in tenths of a degree
|     intin(2) -- End angle in tenths of a degree
|     ptsin(1) -- X coordinate of arc centrepoint
|     ptsin(2) -- Y coordinate of arc centrepoint
|     ptsin(3) -- X coordinate of arc startpoint
|     ptsin(4) -- Y coordinate of arc startpoint
|     ptsin(5) -- X coordinate of arc endpoint
|     ptsin(6) -- Y coordinate of arc endpoint
|     ptsin(7) -- Radius
|     ptsin(8) -- 0

```

```

| For the CIRCLE:-

```

```

|     Contrl(2) -- No. vertices =3
|     Contrl(6) -- Primitive I.D.=4
|     ptsin(1) -- X coordinate of circle centrepoint
|     ptsin(2) -- Y coordinate of circle centrepoint
|     ptsin(3) -- X coordinate of point on circumference

```

```

| ptsin(4)  --   Y coordinate of point on circumference |
| ptsin(5)  --   Radius |
| ptsin(6)  --   0 |
| For the PRINT GRAPHIC CHARACTERS:- |
| Contrl(2) --   No. vertices =1 |
| Contrl(4) --   Number of characters to output |
| Contrl(6) --   Primitive I.D.=5 |
| ptsin(1)  --   X coordinate of start point |
| ptsin(2)  --   Y coordinate of start point |
|
| Output parameters. |
|
| Control array |
| Contrl(3)  --   No. of output vertices=0 |

```

---

GDOS OPCODE 12 - Set Character Height

---

This operation sets the actual character height, width, and the character cell dimensions in device units. (the character dimensions are those of the uppercase 'W'). It returns the nearest dimensions to the request of which it is capable.

Input parameters

Control array

```

| Contrl(1) --   Opcode = 12 |
| Contrl(2) --   Number of vertices =1 |
| Point input array. |
| Ptsin(1)  --   0 |
| Ptsin(2)  --   Requested character height |

```

Output parameters.

```

| Contrl(3) --   No. of vertices = 2 |
| Ptsout(1) --   Actual character width (device units) |
| Ptsout(2) --   Actual character height (device units) |
| Ptsout(3) --   character cell width (device units) |
| Ptsout(4) --   character cell height (device units) |

```

---

GDOS OPCODE 13 - Set Text Direction

---

This operation sets the orientation in which text is written on the device. For example, text may be best written vertically for labelling Y axes. 0 degrees is considered to be the normal orientation of text, with angles increasing in a counterclockwise direction. The driver returns the actual rotation that fits the request best.

## Input parameters

## Control array

Contrl(1) -- Opcode = 13  
 Contrl(2) -- Number of vertices = 0

## Integer input parameters

intin(1) -- Requested angle of rotation of the  
 character baseline in tenths of a degree.  
 intin(2) -- Run of angle (Cos angle \*100)  
 intin(3) -- Rise of angle (Sin angle \* 100)

## Output parameters.

## Control array

Contrl(3) -- No. of output vertices=0  
 Contrl(5) -- 1

## Integer Output parameters

Intout(1) -- Angle of character rotation (in tenths  
 of a degree)

---



---

GDOS OPCODE 14 - Specify colour index value

---

This operation associates a colour index with the colour specified in RGB units. In the minimal case (monochrome) two colour indices (black and white) are required, and a colour device has one index for each colour. This call is only useful on colour devices with palettes.

## Input parameters

## Control array

Contrl(1) -- Opcode = 14  
 Contrl(2) -- Number of vertices = 0

## Integer input parameters

intin(1) -- Colour index  
 intin(2) -- Red colour intensity (0-1000)  
 intin(3) -- Green colour intensity (0-1000)  
 intin(4) -- Blue colour intensity (0-1000)

## Output parameters.

## Control array

Contrl(3) -- No. of output vertices=0

---

---

GDOS OPCODE 15 - Set Polyline linetype

---

This operation sets the type of line drawn in subsequent polyline operations. Refer to the device specifications for the linetypes of which the device is capable.

## Input parameters

## Control array

Contrl(1) -- Opcode = 15  
 Contrl(2) -- Number of vertices = 0  
 Integer input parameters  
 intin(1) -- Requested linestyle

## Output parameters.

## Control array

Contrl(3) -- No. of output vertices=0  
 Intout(1) -- Linestyle selected

---



---

GDOS OPCODE 16 - Set Polyline line Width

---

This operation sets the width of lines drawn in subsequent polyline operations. Refer to the device specifications for the widths of line of which the device is capable.

## Input parameters

## Control array

Contrl(1) -- Opcode = 16  
 Contrl(2) -- Number of vertices = 0  
 Input point array  
 ptsin(1) -- Requested linewidth

## Output parameters.

## Control array

Contrl(3) -- No. of output vertices=0  
 ptsout(1) -- selected Linewidth  
 ptsout(2) -- 0

---



---

GDOS OPCODE 17 - Set Polyline colour index

---

This operation sets the colour of lines drawn in subsequent polyline operations. The colour is the one currently set in the colour index.

## Input parameters

## Control array

Contrl(1)	--	Opcode = 17
Contrl(2)	--	Number of vertices = 0
Intin(1)	--	Requested colour index

## Output parameters.

## Control array

Contrl(3)	--	No. of output vertices=0
intout(1)	--	Colour Index selected

---



---

GDOS OPCODE 18 - Set Polymarker type

---

This operation sets the marker type drawn in subsequent polymarker operations. Refer to the device specifications for the markers of which the device is capable.

## Input parameters

## Control array

Contrl(1)	--	Opcode = 18
Contrl(2)	--	Number of vertices = 0
Intin(1)	--	Requested polymarker type

## Output parameters.

## Control array

Contrl(3)	--	No. of output vertices=0
intout(1)	--	polymarker type selected

---

---

GDOS OPCODE 19 - Set Polymarker height (scale)

---

This operation sets the marker height in subsequent polymarker operations. The driver returns the actual height nearest to the requested one, of which the device is capable.

## Input parameters

## Control array

Contrl(1)	--	Opcode = 19
Contrl(2)	--	Number of vertices =1
Ptsin(1)	--	0
Ptsin(2)	--	requested Polymarker height

## Output parameters.

## Control array

Contrl(3)	--	No. of output vertices=0
Ptsout(1)	--	0
Ptsout(2)	--	actual Polymarker height selected

---



---

GDOS OPCODE 20 - Set Polymarker colour index

---

This operation sets the colour of the markers for subsequent polymarker operations.

## Input parameters

## Control array

Contrl(1)	--	Opcode = 20
Contrl(2)	--	Number of vertices =0
Intin(1)	--	Requested polymarker colour index

## Output parameters.

## Control array

Contrl(3)	--	No. of output vertices=0
Intout(1)	--	Polymarker colour index selected

---

+-----+	
GDOS OPCODE 21 - Set The hardware text font	
-----	
This operation sets a font for subsequent graphics text operations. Fonts are hardware-dependent and range from 1 to the device maximum	
Input parameters	
Control array	
Contrl(1)	-- Opcode = 21
Contrl(2)	-- Number of vertices =0
Intin(1)	-- Requested hardware text font number
Output parameters.	
Control array	
Contrl(3)	-- No. of output vertices=0
Intout(1)	-- hardware font selected
-----	
+-----+	
GDOS OPCODE 22 - Set The Colour Index	
-----	
This operation sets colour index for subsequent text operations. the index ranges from 1 to the device maximum.	
Input parameters	
Control array	
Contrl(1)	-- Opcode = 22
Contrl(2)	-- Number of vertices =0
Intin(1)	-- Requested text colour index
Output parameters.	
Control array	
Contrl(3)	-- No. of output vertices=0
Intout(1)	-- Text Colour index selected
-----	
+-----+	

---

GDOS OPCODE 23 - Set The Interior Fill Style

---

This operation sets the polygon polyfill interior fill style to be used for subsequent polygon fill operations if the device is capable of the style. the index ranges from 1 to the device maximum.

## Input parameters

## Control array

Contrl(1) -- Opcode = 23  
 Contrl(2) -- Number of vertices =0  
 Intin(1) -- Requested fill interior style  
 0=hollow/no fill, 1=solid, 2=halftone 3=hatch

## Output parameters.

## Control array

Contrl(3) -- No. of output vertices=0  
 Intout(1) -- Fill interior style selected

---



---

GDOS OPCODE 24 - Set The Fill Style Index

---

This operation sets the polyfill style index for polyfill operations where the interior style is hatched or patterned. All subsequent polygon fill operations use the style and index. If the style is hatched, various hatch styles are used whereas, if the style is halftone, various grey scale shadings are used.

## Input parameters

## Control array

Contrl(1) -- Opcode = 24  
 Contrl(2) -- Number of vertices =0  
 Intin(1) -- Requested fill style index

## if halftone

1=vertical lines, 2=horizontal lines, 3=+45 degree lines

4= -45 degree lines, 5=horizontal/vertical cross

6= +45 degree

## if hatch

1-6 are halftone patterns: 1=lightest, 6=darkest

## Output parameters.

## Control array

Contrl(3) -- No. of output vertices=0  
 Intout(1) -- Fill style index selected

---

```

+-----+
| GDOS OPCODE 25 - Set The Fill colour Index |
+-----+
| This operation sets the colour index for polyfill operations |
| The RGB values for the actual index can be altered by the |
| 'SET_COLOUR_REPRESENTATION' op. |
|
| Input parameters |
|
|     Control array |
|     Contrl(1)  --  Opcode = 25 |
|     Contrl(2)  --  Number of vertices =0 |
|     Intin(1)   --  Requested fill colour index |
|
| Output parameters. |
|
|     Control array |
|     Contrl(3)  --  No. of output vertices=0 |
|     Intout(1)  --  Fill colour index selected |
+-----+
| GDOS OPCODE 26 - Return colour representation |
+-----+
| This operation returns the requested or actual value of the |
| specified colour index in RGB units. |
| GSX has the capability of allowing the program to specify |
| the exact tone of colour when using devices that use |
| palette mixing to write in a huge range of colours. |
| The RGB values for the actual index can be altered by the |
| 'SET_COLOUR_REPRESENTATION' op. |
|
| Input parameters |
|
|     Control array |
|     Contrl(1)  --  Opcode = 26 |
|     Contrl(2)  --  Number of vertices =0 |
|     Intin(1)   --  Requested colour index |
|     Intin(2)   --  Set or realised |
|                   0= return requested values, 1= return realised values |
|
| Output parameters. |
|
|     Control array |
|     Contrl(3)  --  No. of output vertices=0 |
|     Intout(1)  --  Fill colour index selected |
|     Intout(2)  --  Red intensity (in tenths of percent) |
|     Intout(3)  --  Green intensity (in tenths of percent) |
|     Intout(4)  --  Blue intensity (in tenths of percent) |
+-----+

```

```
-----+
GDOS OPCODE 27 - Return cell array definition
-----+
```

```
Returns the cell array definition of the specified cell. This
call is analogous to the CELL_ARRAY (opcode 10) call. Colour
indices are returned one row at a time, starting from the top
of the rectangular area, proceeding downwards. This call was
an attempt at a pixel-oriented, rather than a vector-oriented
approach to graphics. The call is seldom implemented
```

```
Input parameters
```

```
Control array
```

```
Contrl(1) -- Opcode = 27
Contrl(2) -- Number of vertices =2
Contrl(4) -- Length of colour index array
Contrl(6) -- Length of each row of array
Contrl(7) -- Number of rows in colour index
ptsin(1) -- X coordinate of lower left
ptsin(2) -- Y coordinate of lower left
ptsin(3) -- X coordinate of upper right
ptsin(4) -- Y coordinate of upper right
```

```
Output parameters.
```

```
Control array
```

```
Contrl(3) -- No. of output vertices=0
Contrl(8) -- The number of elements used in each
row of the colour index array
Contrl(9) -- Number of rows used in the colour index
array.
Contrl(10) -- Error flag (0=no errors, 1=a colour
value could not be determined
Intout -- Colour index array stored one row at a
time.
```

---

GDOS OPCODE 28 - Return locator position

---

This call enables the user to specify a location on the screen. There are two modes. In request mode, a graphic cursor appears on the screen, and can be moved by keystroke, mouse, joystick or trackball until the desired position is reached. At this point, a key or button is pressed to terminate the input operation, and the new coordinates are returned. In sample mode, no cursor is displayed, and the status of the locator device is sampled, and returned to the calling program.

## In Request Mode:-

## Input parameters

## Control array

Contrl(1) -- Opcode = 28  
 Contrl(2) -- Number of vertices =1  
 Intin(1) -- Locator device to use (1=keyboard,  
 2=mouse or joystick  
 ptsin(1) -- X coordinate of locator  
 ptsin(2) -- Y coordinate of locator

## Output parameters.

## Control array

Contrl(3) -- No. of output vertices=1  
 Contrl(5) -- Length of intout array (0 if not  
 successful  
 Intout(1) -- The terminator character used to end  
 the operation (eg CR, Tab, space)  
 Ptsout(1) -- Final X coordinate of locator  
 Ptsout(2) -- Final Y coordinate of locator

## In Sample Mode:-

## Input parameters

## Control array

Contrl(1) -- Opcode = 28  
 Contrl(2) -- Number of vertices =1  
 Intin(1) -- Locator device to use (1=keyboard,  
 2=mouse or joystick

## Output parameters.

## Control array

Contrl(3) -- No. of output vertices (1 if change, 0  
 if no change.)  
 Contrl(5) -- Length of intout array (0 if no  
 terminating character, 1 if terminating character  
 Intout(1) -- Any terminator character used to end  
 the operation (eg CR, Tab, space)  
 Ptsout(1) -- Any new X coordinate of locator  
 Ptsout(2) -- Any new Y coordinate of locator

---

```
-----+
GDOS OPCODE 29 - Return value of valuator device
-----+
```

This operation is in two different modes. In request mode, a numeric value is displayed, and can be incremented or decremented by means of keystroke or otherwise until a terminating character is struck. In sample mode, the current value is returned. This status mode, as in the preceeding opcode are only strictly relevant if the input is done under interrupts.

In Request Mode:-

Input parameters

Control array

```
Contrl(1)  -- Opcode = 29
Contrl(2)  -- Number of vertices = 0
Intin(2)   -- Initial value
```

Output parameters.

Control array

```
Contrl(3)  -- No. of output vertices=0
Contrl(5)  -- Length of intout array (1)
Intout(1)  -- Output value
Intout(2)  -- Terminator (ASCII)
```

In Sample Mode:-

Input parameters

Control array

```
Contrl(1)  -- Opcode = 29
Contrl(2)  -- Number of vertices = 0
```

Output parameters.

Control array

```
Contrl(3)  -- No. of output vertices=0
Contrl(5)  -- Length of intout array (0 if nothing
             happened, 1 if valuator changed, 2 if terminator)
Intout(1)  -- New valuator value
Intout(2)  -- Any terminator (ASCII)
```



```

+-----+
| GDOS OPCODE 30 - Return choice device status keys |
+-----+
| This operation typically allows a program to access function |
| keys to enable the program user to make a choice from a menu |
| or prompt. In request mode, the keyboard status is sampled |
| until a key is pressed and a number from 1 to the device |
| maximum is returned. In sample mode, the current status of |
| the function keys is returned but the driver does not wait |
| for a key to be pressed. |
| In Request Mode:- |
| Input parameters |
| |
| Control array |
| Contrl(1) -- Opcode = 30 |
| Contrl(2) -- Number of vertices =0 |
| Intin(1) -- Choice device number (1=function keys, |
| 1+=workstation dependent) |
| |
| Output parameters. |
| |
| Control array |
| Contrl(3) -- No. of output vertices=0 |
| Contrl(5) -- Length of intout array=1 |
| Intout(1) -- Choice number (1-workstation maximum |
| |
| In Sample Mode:- |
| Input parameters |
| |
| Control array |
| Contrl(1) -- Opcode = 30 |
| Contrl(2) -- Number of vertices =0 |
| Intin(1) -- Choice device number (1=function keys, |
| 1+=workstation dependent) |
| |
| Output parameters. |
| |
| Control array |
| Contrl(3) -- No. of output vertices=0 |
| Contrl(5) -- Choice status.(0=nothing happened, 1= |
| sample successful, 2=nonchoice key) |
| Intout(1) -- Any Choice number (1-workstation max) |
| Intout(2) -- Any choice terminator |
+-----+

```

-----+  
GDOS OPCODE 31 - Return string from specified string device

This operation is in one of two modes, request or sample.  
 In Request mode, GSX returns a string typed from the keyboard, waiting for entry and returning when a carriage return is struck. In status mode, the string operation is undertaken if a key has been struck, and terminates as soon as no more keystrokes are available.

## Input parameters

## In Request Mode:-

## Control array

Contrl(1) -- Opcode = 31  
 Contrl(2) -- 0 if not echoing, 1 if echoing.  
 Intin(1) -- string device number (1=keyboard, 1+=workstation dependent)  
 Intin(2) -- Maximum string length  
 Intin(3) -- Echo mode (0=not echo input characters  
 Ptsin(1) -- X coordinate of echo area in echo mode  
 Ptsin(2) -- Y coordinate of echo area if echo mode

## Output parameters.

## Control array

Contrl(3) -- No. of output vertices=0  
 Contrl(5) -- Length of intout array (length of string if successful, 0 if not successful)  
 Intout -- the string

## In Sample Mode:-

## Control array

Contrl(1) -- Opcode = 31  
 Contrl(2) -- 0  
 Intin(1) -- string device number (1=keyboard, 1+=workstation dependent)  
 Intin(2) -- Maximum string length

## Output parameters.

## Control array

Contrl(3) -- No. of output vertices=0  
 Contrl(5) -- Length of intout array (length of string if successful, 0 if not successful)  
 Intout -- accumulated characters

```

+-----+
| GDOS OPCODE 32 - Set writing mode |
+-----+

```

```

| This operation specifies the way that lines, pixels, or text |
| are drawn on the screen. There are four possible ways of |
| doing this where a dotted line or pattern are involved. |
| Replace mode writes like an opaque ink, and the background |
| is used for the 'dark' part of the pattern. Everything |
| underneath is completely covered. Transparent mode is |
| similar except that the 'dark' part of the pattern is not |
| altered from what was there before. XOR mode reverses the |
| bits representing the colour, and erase mode draws in the |
| background colour, leaving the 'dark' part of the pattern |
| as it was. |

```

```

| Input parameters |

```

```

|     Control array |

```

```

| Contr1(1) --      Opcode = 32 |
| Contr1(2) --      Number of vertices =0 |
| Intin(1)  --      Writing mode (1=replace, 2=transparent |
|              3=XOR, 4=erase) |

```

```

| Output parameters. |

```

```

|     Control array |

```

```

| Contr1(3) --      No. of output vertices=0 |
| Intout(1) --      Writing mode selected |
+-----+

```

```

+-----+
| GDOS OPCODE 33 - Set input mode |
+-----+
| This operation sets the input mode (sample or request) for |
| the specified logical input device. In request modes, GSX |
| waits until a response has been completed before returning |
| whereas sample modes merely return the status of the device |
| at the time. |
| |
| Input parameters |
| |
| Control array |
| Contr1(1) -- Opcode = 33 |
| Contr1(2) -- Number of vertices =0 |
| Intin(1) -- Logical input device (1=locator, 2= |
| valuator, 3=choice, 4=string) |
| Intin(2) -- Input mode (1=request, 2=sample) |
| |
| Output parameters. |
| |
| Control array |
| Contr1(3) -- No. of output vertices=0 |
| Intout(1) -- Input mode selected. |
+-----+

```

## CHAPTER 10 – Some CP/M Programming Languages

- 10.1 – Astec C
- 10.2 – Small C
- 10.3 – BDS C
- 10.4 – ALGOL/M
- 10.5 – CBASIC
- 10.6 – MBASIC
- 10.7 – BASCOM
- 10.8 – PASCAL MT+

In this chapter, we give quick reference guides to some of the more popular CP/M languages. They are no substitute for the proper documentation but are intended to provide quickly accessible information for running or compiling programs. We start with the three most popular dialects of 'C'. Astec C is popular because it is almost complete and standard, and is compatible with Astec compilers for other operating systems and CPUs. Small C is popular because it is free and comes with the source, so it can easily be altered. BDS C, which follows, was the first available C for CP/M, and it is very fast to compile. It represents very good value for money. Leaving C we pass on to Algol/M. This, like BASIC-E, has a claim to be part of the CP/M scene, as it was developed at the same time in the same place. It deserves to become better known, particularly as a learning tool. It is in the public domain, and would be useful in an educational setting. After Algol, we come to the premier CP/M BASICs. One cannot avoid BASIC on a micro, and Microsoft Basic (MBASIC and BASCOM) are the standard for both CP/M and MSDOS. (Mallard Basic is a superset of MBASIC). CBASIC is Digital Research's answer to Microsoft Basic. The relative merits of the two dialects is a matter for debate, but they both remain popular. Lastly, we cover Pascal MT+, because this is Digital Research's own Pascal compiler. It is, in its latest reincarnation, reasonably cheap and robust but it has been eclipsed by both Turbo Pascal and Propascal in popularity.

### 10.1 Astec C

Astec C is a complete C for 8080 or Z80 processors. It is compatible with almost all of the features of UNIX System 5, and has compilers for the 8086, and 68000 chips. The compiler is started by:-

**A>CC -x Filename**

where filename is a CP/M filename and -x is one or more options.

If the filename has not filetype specified, then a filetype of '.C' is assumed. if the filename is preceded by a number and a backslash, it is taken to be the user area to search on for the file. Options can include:-

- d SYMBOL  
Defines a symbol for the preprocessor.
- i  
Defines an area to be searched for files in the `#include` statement.
- o FILENAME  
Specifies an alternate output file.
- s  
Cause a search for undefined structure members.
- t  
Include C source statements in the assembly code as comments.

```

-f          Forces in-line frame allocation.
-p          Send error messages to the printer.
-q          Convert automatic variables to statics for efficiency.
-u          Convert uninitialized global variables into externs.
-m          Generate code for the microsoft assembler. M80
-r          Generate code for the digital research assembler RMAC

-e nnn      Specifies the size of the expression table.
-l nnn      Specifies the size of the local table.
-y nnn      Specifies the maximum number of cases allowed in a switch.
-z nnn      Specifies the size of the table for literal strings.

```

If all goes well in the compilation (in later versions of ASTEC CII the error messages are descriptive), then the assembly source produced by the compiler needs to be assembled.

If one uses the ASTEC assembler (one has the option of using M80 or RMAC), this is invoked by typing:-

```
A>AS -x filename.typ
```

where filename.typ is a valid CP/M filename. If the type is left out, the assembler assumes the filetype '.ASM'. The object file which it produces has the filetype '.O', with the filename taken from the command line. -x is optional, and can be one or more conditional switches as follows:-

```

-o filename.typ      sends the object file to the specified filename
-l                  sends a listing to a file derived from the command line file with
                    the filetype '.LST'.
-zap                delete the assembly file after successful assembly.

```

After successful assembly, the linker is used to link in the object file with other modules and whatever library modules are required from the runtime library.

```
A>LN -x file1.o....filen.o lib1.lib...libn.lib
```

where file1.o....filen.o is one or more object files and lib1.lib...libn.lib is one or more library files. For example:-

```
A>ln foo.o c.lib
```

```
or
```

```
A>ln foo.o zot.o zip.o my.lib m.lib c.lib
```

linker options include:-

```

-f filename.typ      Reads command arguments from filename.typ
-l libname           Searches the library file libname.lib
-o filename          Writes the executable file to filename.COM.
-t                  Generate a symbol table file.
-v                  Be verbose.
-b addr             Set the base address of the program to addr (in hex).
-c addr             Set the beginning of the code segment to addr (in hex)
-d addr             Set the beginning of the data segment to addr (in hex)
-u addr             Set the beginning of the uninitialized segment to addr (in hex)
-r                  Create a symbol table to be used in linking overlays.
+c nnnn             Reserve nnnn bytes at the end of the programs code segment.
+d nnnn             Reserve nnnn bytes at the end of the programs data segment.

```

## 10.2. Small C

This compiler can compile only a subset of the standard C language. It was written by Ron Cain and originally published in Dr. Dobbs Journal of Computer Calisthenics and Orthodontia, No.45. It has had many improvements since then and there are versions with 'printf' and floating point maths. There exist several versions, which all work rather differently.

The compiler inputs a program written in Small C from a file and produces a version of the program in assembler language mnemonics which can be assembled and run in a CP/M computer. The original run time library has been supplemented in most current versions with some additional routines which use CP/M I/O facilities. The compiler produces code for an absolute assembler such as ASM.COM and so the libraries are 'included' at compile time is divided into modules so that routines that are not needed for particular applications can be omitted. There is no linkage and no library '.REL' file.

All variables must be declared. If they are declared outside a function they are global. Variables and function names should consist of not more than 8 alpha-numeric characters. Global variables and function names appear as labels in the assembler language file produced by the compiler, so they must satisfy any special requirements of the assembler that is to be used.

Only four data types are permitted, char, int, pointers and arrays (one dimensional):-

All the unary and binary operators are implemented but the && and || logical operators of standard C are missing. (The bitwise & and | can be used with care)

The following preprocessor Statements are implemented

<code>#define name string</code>	Replace name by string throughout the program
<code>#include filename</code>	Insert named file at this point in the program
eg. <code>#include CRUN.LIB</code>	Included files may not contain <code>#include</code> statements.
<code>#asm ... #endasm</code>	This allows assembly language to be included in a program. Anything between <code>#asm</code> and <code>#endasm</code> is passed straight to the O/P file.

Small C programs should start with a statement: `"#include CRUN.LIB"`. The run time routines in CRUN.LIB include an initial section which will set up the stack and then call the function "main()" which must exist in all C programs. The group of routines in CRUN.LIB are essential in all Small C programs, except for the "getp" and "putp" functions. The other three libraries can be `#included` if required. CONIO.LIB contains functions that will normally be needed for CP/M character and string I/O from the console, FILE.LIB (which assumes CONIO.LIB is also present) can be `#included` if files are going to be used. NUMIO.LIB contains some routines in C for inputting and outputting decimal and hexadecimal numbers.

Only one input and one output file may be open at a time. CON: and LST: are acceptable O/P names. After compilation the stack pointer setting and the ORG pseudo-op can be set to suit a particular application if required.

As SMALL C exists in several versions, it is not easy to give instructions for use. Generally speaking, one types the name of the compiler, and it interrogates the user for the required parameters.

Small C does not produced well-optimised code, but it is usable and in the public domain. The source (in small C) is available too.



## 10.3 BDS C

BDS C was the first good C compiler to be available to run under CP/M. It has attracted a large following despite its rather unconventional features. C is an important language, but, until recently, was not efficient enough to be effective in an 8-bit environment. Software technology has now made more efficient compilers possible, but there is still a problem in producing really compact code on a microprocessor that was not really designed for high-level languages. BDS C is very fast to compile, possibly because it bypasses the stage of producing assembler source from C source. BDS C has a non-standard run-time library and the floating point module is an afterthought. Its strength is the fact that it supplies the library source and it ships a large number of sample programs to illustrate the way that the program works.

To compile a source file using BDS C, one can use the supplied SUBMIT file C.SUB. If the SUBMIT file is used, it is engaged as follows :-

**SUBMIT C filename**

where 'filename' is the name of 'filename.C', the file to compile. Note that the user is NOT to type filename.C, but is just to type filename.

As execution of the two passes of the compiler and the linker proceeds, the user will be given the chance to abort processing at various critical points in the process by the execution of the ABORTSUB program. If an error has occurred during the previous processing, ABORT when this program is executed.

BDS C is not, as is customary, a compiler called CC.COM. Because C loads the entire source file into memory in one gulp, the compilation is broken up into two phases (not "passes", strictly). The two phases end up taking about 8 passes to actually implement the compilation, maximizing the amount of memory available for the source file. CC1, the first half of the compiler, accepts a C source file with any filename and extension (say, "super.c") and writes out a temporary file (with the same filename and extension ".CCI") containing a symbol table and an encoded form of the source code.

Unlike most compilers, the file extension ".C" is NOT assumed for the input file, so saying "super" for "super.C" would not work.

If the source file name is preceded by a disk designation, then the input is taken from the specified disk and the output is also written to that disk.

If any errors are detected during CC1, the output file will not be written.

One can specify options in the command line, preceded by a dash (-):

s causes undeclared identifiers to be implicitly declared as integer variables, wherever possible.

hex digit (4-f) sets symbol table size to the specified value (in K bytes); default is 8 (5 for versions x.xT.)

For example:-

A>CC1 CHESS.C -s6      Suppresses errors for undefined variables and sets symbol table size to 6K bytes;

A>cc1 BASIC.C -e               Sets symbol table size to 14K bytes. Note that the option list must contain no blanks.

A>cc1 b:LINK.C                Takes the source file from disk B and writes the '.CCI' file to disk B (regardless of what the currently logged disk is.)

The second phase is done by CC2 This is the second half of the compiler. CC2 accepts a ".CCI" file as input, and writes out a ".CRL" file if no errors are detected. (CRL stands for 'C ReLocatable' and it is a non-standard relocatable format)

If all goes well, writing out of the CRL file is followed by deletion of the "CCI" file, and compilation is complete.

The CRL file must be linked in with other modules and the C run-time module, using CLINK. Digital Research's LINK cannot be used because the CRL format is not the same as the standard Microsoft '.REL' format. CLINK links a "main" function from some CRL file together with C.CCC (for common system subroutines) and any subordinate functions which "main" may require (from perhaps many CRL files). A successful linkage causes a ".COM" file to be generated. This can now be executed.

The first argument on the command line after a LINK command must be the name of a CRL file containing a "main" function. If the name is specified with an extension, then that extension is interpreted specially as indicating which disks are to be involved in the operation (this is akin to the mechanism ASM uses to determine source and destination disks.)

For example, if the first argument to CLINK were given as:

A>clink super.bc  
then CLINK would interpret the "b" in ".bc" as specifying the disk on which "DEFF.CRL" and "C.CCC" are to be found, and the "c" in ".bc" as specifying which disk the '.COM' file is to be written to. Both of these values, if omitted, default to the currently logged in disk. The first argument may also be preceded by a disk designation, to specify where all '.CRL' files are to be searched for (by default). For example, the command  
A>clink b:zot.ac  
tells CLINK to get C.CCC and DEFF.CRL from disk A; to write the output file to disk C; and to find ZOT.CRL on disk B.

Any other CRL files to search may also be specified on the command line (WITHOUT their '.CRL' suffixes), causing those to be searched in the order specified. The default disk to search will be the same disk from which the original CRL file was taken; this default can be overridden by specifying an explicit disk designation for any appropriate CRL file name needing it. For example,

A>clink c:super.bb bar a:zot fraz  
causes disk C to be searched for the files super.CRL, BAR.CRL and FRAZ.CRL, while disk A would be searched to find ZOT.CRL. Disk B is where CLINK would expect DEFF.CRL and C.CCC to reside, and the output would go to disk B also.

When all given CRL files have been searched, CLINK will automatically search DEFF.CRL.

If there are still some unresolved references, then CLINK will ask for

input from the keyboard to try resolving them.

There are also several options which may be specified on the command line. Each option must be preceded by a dash (-); the space between options and their argument (if needed) is optional. The presently supported options are:

- s Prints out load statistics;
- t nnnn Reserves location nnnn (hex) and above for user; default is to reserve no space. What this really does is to cause the first op in the object file to be lxi sp,nnnn instead of lxi sp,bdos.
- o name Causes the '.COM' file generated to have the given name. Default is the name of the first '.CRL' file given (the one with the "main" function.)
- e xxxx Sets start of data area to address xxxx, to maintain consistency between several separate '.COM' files when chaining (via the library function "exec") is used.
- c Specifies that the '.COM' file is to be chained to from another '.COM' file. If the resultant '.COM' file is invoked directly from CP/M instead of via the "exec" function, then ARGV & ARGV processing is suspended, since upon being chained to you wouldn't want ARGV & ARGV processing to take place. Note that if you use this option, you should also use the -e option to set the data area address equal to that of the chaining '.COM' file.

#### Examples:

A>clink super bar gets "main" from the file super.CRL, searches for needed functions first in super.CRL and then, if needed, in BAR.CRL and DEFF.CRL. All files are assumed to reside on the currently logged in disk.

A>clink b:ihftp belle -s searches for IHTFP.CRL and BELLE.CRL on disk B; prints a statistics summary when linkage is complete. The files DEFF.CRL and C.CCC are assumed to reside on the currently logged in disk; output also goes to the currently logged in disk.

A>clink b:ihftp.aa -s belle -o zot is the same as the last example except: the output file is called ZOT.COM, DEFF.CRL and C.CCC are assumed to reside on A, and output goes to A.

A>clink stoned -t7000 -s sets top of memory to 7000h and prints out load statistics. Current disk used for everything.

DEFF.CRL: This file contains the standard function library... all 60+ functions worth. See the BDS C User's Guide for documentation on these functions.

C.CCC: The run-time skeleton file, containing code for processing the command line (generating argc and argv, for you UNIX lovers), room for file I/O

buffers, some math subroutines, etc.

#### 10.4 Algol/M

ALGOL/M is a version of ALGOL-60. It is an excellent language with which to learn programming, particularly if funds are short. It is in the public domain and is contained in Volume 28 of the CP/M User group library. It has never been as popular as BASIC-E, though it is every bit as good. It is similar in many ways, including its organisation as a pseudo-compiler. It was written at the Naval Postgraduate school at the same time as BASIC-E and shows the Kildall influence. Source code is compiled into an intermediate code which is then interpreted by a run-time monitor.

To compile the source code, type:-

ALGOLM filename Soption

where 'filename' is the name of the file 'filename.ALG' which contains the source of the ALGOL/M program, and where Soption is one of the following:-

SA	Generate a listing at the terminal
SE	Set Trace Mode for execution under RUNALG
SAE	Do Both of the above

Upon executing this command, ALGOL/M will compile this source program into the pseudo code file 'filename.AIN' which can then be executed by typing --

RUNALG filename

where 'filename' is the name of the file 'filename.INT' which contains the ALGOL/M pseudo code.

Although ALGOL-M was modelled on ALGOL-60, no attempt was made to make it a formal subset of ALGOL-60. This was done intentionally in order to provide a language which would be better suited to the microcomputer environment. However, the basic structure of ALGOL-M is similar enough to ALGOL-60 to allow simple conversion of programs from one language to the other. This was considered particularly important in view of the fact that one of the standard publication languages is ALGOL-60. Therefore, there exists a large source of applications programs and library procedures which can be simply converted to execute under ALGOL-M. ALGOL has been superceded in the main by Pascal, but it has merits that Pascal lacks.

ALGOL-M supports three types of variables: integers, decimals, and strings. As the decimals are in BCD (Binary Coded Decimal), this gives a good precision for commercial work. Integers may be any value between -16,383 and +16,383. Decimals may be declared with up to 18 digits of precision and strings may be declared as long as 255 characters. The default precision for decimals is ten digits and the default length for strings is ten characters. Decimal and string variable lengths may be integer variables which can be assigned actual values at run-time. Another form of declaration in ALGOL-M is the array declaration. Arrays may have up to 255 dimensions with each dimension ranging from 0 to +16,383. The maximum 8080 microprocessor address space of 64k bytes limits practical array sizes to something smaller than the maximum. Dimension bounds may be integer variables with the actual values assigned at run-time. Arrays may be of type integer, decimal or string.

Integer and binary coded decimal arithmetic are supported under ALGOL-M. Integers may be used in decimal expressions and will be converted to decimals at run-time. The integer and decimal comparisons of less-than (<), greater-than (>), equal-to (=), not-equal-to (<>), less-than-or-equal-to (<=), and greater-than-or-equal-to (>=) are provided. Additionally, the logical operators AND, OR and NOT are available.

ALGOL-M control structures consist of BEGIN, END, FOR, IF THEN, IF THEN ELSE, WHILE, CASE and GOTO constructs. Function and procedure calls are also used as control structures. ALGOL-M is a block structured language with a block normally bracketed by a BEGIN and an END. Blocks may be nested within other blocks to nine levels. Variables which are declared within a block can only be referenced within that block or a block nested within that block. Once program control proceeds outside of a block in which a variable has been declared, the variable may not be referenced and, in fact, run-time storage space for that variable no longer exists.

Functions, when called, return an integer, decimal or string value depending on the type of the function. Procedures do not return a value when called. Both functions and procedures may have zero or more parameters which are call by value and both may be called recursively.

The ALGOL-M WRITE statement causes output to the console on a new line. The desired output is specified in a write list which is enclosed in parentheses. String constants may be used in a write list and are characterized by being enclosed in quotation marks. Any combination of integer, decimal and string variables or expressions may also be used in a write list. A WRITEON statement is also available which is essentially the same as the WRITE statement except that output continues on the same line as the output from a previous WRITE or WRITEON statement. When a total of 80 characters have been written to the console, a new line is started automatically. A TAB option may also be used in the write list which causes the following item in the write list to be spaced to the right by a specified amount.

Console input is accomplished by the READ statement followed by a read list of any combination of integer, decimal and string variables enclosed in parentheses. If embedded blanks are desired in the input for a string variable, the console input must be enclosed in quotation marks. A READ statement will result in a halt in program execution at run-time until the input values are typed at the console and a carriage return is sent. If the values typed at the console match the read list in number and type, program execution continues. If an error as to number or type of variables from the console occurs, program execution is again halted until values are re-entered on the console.

ALGOL-M programs may read data from, or write data to, one or more disk files which may be located on one or more disk drives. When file input or output is desired, the appropriate READ or WRITE statement is modified by placing a filename identifier immediately after READ or WRITE. The actual name of the file may be assigned to the file name identifier when the program is written or it may be assigned at run-time. Various disk drives are referenced by the letters A to Z. A specific drive may be specified by prefixing the actual file name with the desired drive letter followed by a colon. Additionally, if random file access is desired, the file name identifier may be followed by a comma and an integer constant or variable. This integer value specifies the record within the file which is to be used for input/output.

Prior to the use of a file name identifier in a READ or WRITE statement, the file name identifier must appear in a file declaration statement. The file name identifier can only be referenced within the same block (or a lower block) as the file declaration. Files are normally treated as unblocked sequential files. However, if blocked files are desired, the record length may optionally be specified in brackets after the file name identifier in the file declaration statement.

The ALGOL/M Reserved Words are:-

AND	ARRAY	BEGIN	CASE
CLOSE	DECIMAL	DO	ELSE
END	FILE	FUNCTION	GO
GOTO	IF	INTEGER	NOT
OF	ONENDFILE	OR	PROCEDURE
READ	STEP	STRING	TAB
THEN	TO	UNTIL	WHILE
WRITE	WRITEON		

The ALGOL-M Compiler Error Messages are:-

AS	Function/Procedure on left hand side of assignment statement.
BP	Incorrect bound pair subtype (must be integer).
DE	Disk error; no corrective action can be taken in the program.
DD	Doubly declared identifier, label, variable etc.
FP	Incorrect file open statement.
IC	Invalid special character.
ID	Subtypes incompatible (decimal values can not be assigned to integer variables).
IO	Integer overflow.
IT	Identifier is not declared as a simple variable or function.
NG	No ALG file found.
NI	Subtype is not integer.
NP	No applicable production exists.
NS	Subtype is not string.
NT	For clause, Step expression, Until clause expressions are not of the same subtype. (must all be integer or decimal).
PC	Number of parameters in procedure call does not match the number in the procedure declaration.
PD	Undeclared parameter.
PM	Parameter type does not match the declared type.
SO	Stack overflow.
SI	Array subscript is not of subtype integer.
TD	Subtype has to be integer or decimal.
TM	Subtypes do not match or are incompatible.
TO	Symbol table overflow.
TS	Undeclared subscripted variable.
UD	Undeclared identifier.
UF	Undeclared file/function.

UL Undeclared label.  
 UP Undeclared procedure.  
 US Undeclared simple variable.  
 VO Varc table overflow. Possibly caused by too many  
 long identifiers.

The ALGOL-M Run-Time Error and Warning Messages are :-

ERROR Messages

AB Array subscript out of bounds.  
 CE Disk file close error.  
 DB Input field length is larger then the buffer size.  
 DW Disk file write error.  
 ER Variable block size write error.  
 IO Integer overflow(integer value greater than 16383).  
 IR Record number incorrect or random file is not initialized.  
 ME Disk file creation error.  
 NA No AIN file found on directory.  
 OV Decimal register overflow during arithmetic operation/  
 load.  
 RE Attempt to read past end of record on blocked file.  
 RU Attempt to random access a non-blocked file.  
 SK Stack overflow(no more memory available).

WARNING Messages

AZ Attempt to allocate null decimal or string,  
 system defaults to 10 digits/characters.  
 DO Decimal overflow during store operation. The value of  
 the variable is set to 1.0 and execution continues.  
 The variable's allocation size should be increased in  
 it's declaration statement.  
 DI Disk file variable format error.  
 DZ Decimal division by zero, result is set to 1.0.  
 EF End of file on read.  
 IA Integer addition/subtraction over/under flow  
 result is set to 1.  
 II Invalid Console Input. Try input again.  
 IR Record number incorrect or random file is not initialized.  
 IZ Integer division by zero. Divisor set to 1 and  
 division is completed.  
 NX Negative exponential. Exponentiation not done.  
 SO Characters lost during string store.

## 10.5 CBASIC

CBASIC is a pseudocompiler BASIC which may be executed on any floppy disk based CP/M system having at least 20K bytes of memory. In order to make the best use of the power and flexibility of CBASIC, a dual floppy disk system and at least 32K of memory is recommended. If CBASIC is executed in a system smaller than 20K, a CP/M LOAD ERROR may occur. CBASIC is an upgraded form of BASIC-E, and readers are referred to that chapter.

The CBASIC system consists of two programs -- CBASIC and CRUN. CBASIC is the compiler, and CRUN is the run-time interpreter. In a typical CBASIC session, the user will write the program using a text editor, compile it using CBASIC (with the \$B option to suppress listing), and run it using CRUN.

CBASIC2 is an upward-compatible (in terms of source code) extension of the original CBASIC. In addition to all of the features of the original CBASIC, CBASIC2 adds the following:

1. Integer variables
2. Chaining with common variables
3. Additional pre-defined functions
4. Cross Reference capability

Note that CBASIC2 is upward-compatible with CBASIC only in terms of the source code files. An INT file created under CBASIC will not execute with the Version 2 Run-Time Monitor (CRUN2).

Compiler toggles are a series of switches that can be set when the compiler is executed. The toggles are set by typing a dollar-sign (\$) followed by the letter designations of the desired toggles starting one space or more after the program name on the command line. Toggles may only be set for the compiler.

Examples of compiler toggles and invocation forms are --

```
CBAS2 INVENTORY $BGF
B:CBAS2 A:COMPARE $GEC
CBAS2 PAYROLL $B
CBAS2 B:VALIDATE $E
```

## CBASIC Compiler Toggles

- TOGGLE B: Suppresses the listing of the program on the console during compilation. If an error is detected, the source line with the error and the error message will be printed even if Toggle B is set. Toggle B does not affect listing to the printer (Toggle F) or disk file (Toggle G).  
Toggle B is initially off.
- TOGGLE C: Suppresses the generation of an INT file. Engaging this toggle will provide a syntax check without the overhead of writing the intermediate file.  
Toggle C is initially off.
- TOGGLE D: Suppresses translation of lower-case letters to upper-case. For example, if Toggle D is on, 'AMT' will not refer to the same variable as 'amt'.  
Toggle D is initially off.
- TOGGLE E: Causes the run-time program (CRUN2) to accompany any



error messages with the CBASIC line number in which the error occurred. Toggle E must be set in order for the TRACE option (see section 13.4 of the manual) to work. Toggle E is initially off.

TOGGLE F: Causes the compiler output listing to be printed on the LST: device in addition to the system console.

Toggle F is initially off.

TOGGLE G: Causes the compiler output listing to be written to diskette. The file containing the compiler listing has the same name as the '.BAS' file, but its type is '.LST'.

Normally the disk listing will be placed on the same source drive as the source file. The operator may select another drive by specifying the desired drive, enclosed in parens, following the G toggle; for example, CBAS2 B:TAX \$(A:) extracts the source from drive B: and sends the listing to drive A:.

Toggle G is initially off.

#### Text Messages

NO SOURCE FILE: <FILENAME>.BAS

The compiler could not locate a source file used in either a CBASIC command or an INCLUDE directive.

PROGRAM CONTAINS n UNMATCHED FOR STATEMENT(S)

There are n FOR statements for which a NEXT could not be found.

PROGRAM CONTAINS n UNMATCHED WHILE STATEMENT(S)

There are n WHILE statements for which a WEND could not be found.

WARNING: INVALID CHARACTER IGNORED

The previous line contains an invalid ASCII character; this character is ignored by the compiler, and a question mark is printed in its place.

OUT OF DISK SPACE

The compiler has run out of disk space while attempting to write either the INT or LST files.

OUT OF DIRECTORY SPACE

The compiler has run out of directory entries while attempting to create or extend a file.

DISK ERROR

A disk error occurred while trying to read or write to a disk file.

INCLUDE NESTING TOO DEEP NEAR LINE n

An include statement near line n in the source program

exceeds the maximum level of nesting of source files.

## 2-Letter Error Codes

BF -- Branch into Function  
A branch into a multiple line function from outside was attempted.

BN -- Bad Number  
An invalid numeric constant was encountered.

CE -- Close Error  
The intermediate (.INT) file could not be closed.

CI -- Close Include  
An invalid file name in an %INCLUDE statement.

CS -- COMMON Statement error  
A COMMON statement which was not the first statement in the program was detected.

CV -- COMMON Variable error  
An improper reference to a subscript variable in a COMMON statement.

DE -- Disk Error  
A disk error occurred while trying to read the '.BAS' file.

DF -- Disk Full  
There was no space on the disk or the disk directory was full. The '.INT' file was not created.

DL -- Duplicate Line number  
The same line number was used on two different lines. Other compiler errors may cause a DL error message to be printed even if duplicate line numbers do not exist.

DP -- Defined Previously  
A variable in a DIM statement was previously defined.

EF -- Exponential Format  
A number in exponential format was input with no digits following the E.

FA -- Function Attribute  
A function name appears on the left side of an assignment statement but is not within that function.

FD -- Function Definition  
A function name that has been previously defined is being redefined in a DEF statement.

FE -- FOR Error  
A mixed mode expression exists in a FOR statement which the compiler cannot correct.

FI -- FOR Index	An expression which is not an unsubscripted numeric variable is being used as a FOR loop index.
FN -- Function parameter Number	A function reference contains an incorrect number of parameters.
FP -- Function Parameter type	A function reference parameter type does not match the parameter type used in the function's DEF statement.
FU -- Function Undefined	A function has been referenced before it has been defined.
IE -- IF Expression	An expression used immediately following an IF evaluates to type string. Only type numeric is permitted.
IF -- In File	A variable used in a FILE statement is of type numeric where type string is required.
IP -- Input Prompt	An input prompt string is not surrounded by quotes.
IS -- Invalid Subscript	A subscripted variable was referenced before it was dimensioned.
IT -- Invalid Toggle	An invalid compiler directive was encountered.
IU -- Invalid Use	A variable defined as an array is used with no subscripts.
MF -- Mixed Format	An expression evaluates to type string when type numeric is required.
MM -- Mixed Mode	Variables of type string and type numeric are combined in the same expression.
MS -- Mixed String	A numeric expression was used where a string expression is required.
ND -- No DEFFN	A FEND statement was encountered without a corresponding DEF.
NI -- NEXT Index	A variable referenced by a NEXT statement does not match the variable referenced by the associated FOR statement.

NU -- NEXT Unexpected  
A NEXT statement occurs without an associated FOR statement.

OF -- Out of Function  
A branch out of a multiple line function from inside the function was attempted.

OO -- ON Overflow  
More than 25 ON statements were used in the program.

PM -- ??  
A DEF statement appeared within a multiple line function. Functions may not be nested.

SE -- Syntax Error  
The source line contains a syntax error.

SF -- SAVEMEM File  
A SAVEMEM statement uses an expression of type numeric to specify the file to be loaded. This expression must be a string. Possibly the quotation marks were left off a string constant.

SN -- Subscript Number  
A subscripted variable contains an incorrect number of subscripts.

SO -- Syntax Overflow  
The expression is too complex and should be simplified and placed on more than one line.

TO -- Table Overflow  
The program is too large for the system. The program must be simplified or the system size increased.

UL -- Undefined Line number  
A line number that does not exist has been referenced.

US -- Undefined String  
A string has been terminated by a carriage return rather than quotes.

VO -- Variable Overflow  
Variable names are too long for one statement.  
This should not occur.

WE -- WHILE Error  
The expression immediately following a WHILE statement is not numeric.

WU -- WHILE Undefined  
A WEND statement occurs without an associated WHILE statement.

Two textual run-time error messages are presented by CRUN:

NO INTERMEDIATE FILE

A file name was not specified with the CRUN command, or no file of type '.INT' and the specified file name was found on disk.

IMPROPER INPUT - REENTER

This message occurs when the fields entered from the console do not match the field specified in the INPUT statement. This can occur when field types do not match or the number of fields entered is different from the number of fields specified. All fields specified by the INPUT statement must be reentered.

2-Letter Warning Codes

DZ -- Divide by Zero

A number was divided by zero. The result is set to the largest valid CBASIC number.

FL -- Field Length

A field length greater than 255 bytes was encountered during a READ LINE. Only the first 255 characters of the record are retained.

LN -- Logarithm error

The argument given in the LOG function was zero or negative. The value of the argument is returned.

NE -- NEgative number

A negative number was specified following the raise to a power operator (^). The absolute value is used in the calculation.

OF -- OverFlow

A calculation produced a number too large. The result is set to the largest valid CBASIC number.

SQ -- Square root error

A negative number was specified in the SQR function. The absolute value is used.

CBASIC Run-time Errors.

AC -- AsCii error

The string used as the argument in an ASC function evaluated to a null string.

BN -- BUFF Number

The value following the BUFF option in an OPEN or CREATE statement is less than 1 or greater than 52.

CC -- Chain Code

A chained program's code area is larger than the main program's code area.

CD -- Chain Data  
A chained program's data area is larger than the main program's data area.

CE -- Close Error  
An error occurred upon closing a file.

CF -- Chain Function  
A chained program's constant area is larger than the main program's constant area.

CP -- Chain Var Storage  
A chained program's variable storage area is larger than the main program's variable storage area.

CS -- Chain SAVEMEM  
A chained program reserved a difference amount of memory with a SAVEMEM statement than the main program.

CU -- Close Undefined file  
A close statement specified a file number that was not active.

DF -- Defined File  
An OPEN or CREATE was specified with a file number that was already active.

DU -- Delete Undefined file  
A DELETE statement specified a file number that was not active.

DW -- Disk Write error  
An error occurred while writing to a file. This occurs when either the directory or the disk is full.

EF -- End of File  
A read past the end of file occurred on a file for which no IF END statement has been executed.

ER -- Error in Record  
An attempt was made to write a record of length greater than the maximum record size specified in the associated OPEN, CREATE, or FILE statement.

FR -- File Rename  
An attempt was made to rename a file to an existing file name.

FT -- File Toggle  
A FILE statement was executed when 20 files were already active.

FU -- File Undefined  
An attempt was made to read or write to a file that was not active.

IF -- Invalid File name

A file name was invalid.

IR -- Invalid Record number

A record number less than one was specified.

IV -- Invalid Version

An attempt was made to execute an INT file created by a Version 1 Compiler.

IX -- ??

A FEND statement was encountered prior to executing a RETURN statement.

LW -- Line Width

A line width less than 1 or greater than 133 was specified in an LPRINTER WIDTH statement.

ME -- MAKE Error

An error occurred while creating or extending a file because the disk directory was full.

MP -- MATCH Parameter

The third parameter in a MATCH function was zero or negative.

NF -- Number of FILE

The file number specified was less than 1 or greater than 20.

NM -- No Memory

There was insufficient memory to load the program.

NN -- No Number field

An attempt was made to print a number with a PRINT USING statement but there was not a numeric data field in the USING string.

NS -- No String field

An attempt was made to print a string with a PRINT USING statement but there was not a string field in the USING string.

OD -- Overflow Data

A READ statement was executed with no DATA available.

OE -- OPEN Error

An attempt was made to OPEN a file that didn't exist and for which no IF END statement had been previously executed.

OI -- ON Index

The expression specified in an ON ??? GOSUB or an ON ??? GOTO statement evaluated to a number less than 1 or greater than the number of line numbers contained in the statement.

OM -- Overflow Memory

The program ran out of memory during execution.

QE -- Quote Error  
An attempt was made to PRINT to a file a string containing a quotation mark.

RB -- Random BUFF  
Random access was attempted to a file activated with the BUFF option specifying more than 1 buffer.

RE -- READ Error  
An attempt was made to read past the end of a record in a fixed file.

RG -- RETURN with no GOSUB  
A RETURN occurred for which there was no GOSUB.

RU -- Random Undefined  
A random read or print was attempted to other than a fixed file.

SB -- SuBscript  
An array subscript was used which exceeded the boundaries for which the array was defined.

SL -- String Length  
A concatenation operation resulted in a string of more than 255 bytes.

SO -- SAVEMEM  
The file specified in a SAVEMEM statement could not be located on the referenced disk.

SS -- SubString error  
The second parameter of a MID\$ function was zero or negative.

TF -- Too many Files  
An attempt was made to have more than 20 active files simultaneously.

TL -- TAB Length  
A TAB statement contained a parameter less than 1 or greater than the current line width.

UN -- Undefined edit string  
A PRINT USING statement was executed with a null edit string.

WR -- WRite error  
An attempt was made to write to a file after it had been read, but before it had been read to the end of the file.

CBASIC Reserved Words

ABS	AND	AS	ASC	ATN
BUFF	CALL	CHAIN	CHRS	CLOSE



COMMAND\$	COMMON	CONCHAR%	CONSOLE	CONSTAT%
COS	CREATE	DATA	DEF	DELETE
DIM	ELSE	END	EQ	EXP
FEND	FILE	FOR	FRE	GE
GO	GOSUB	GOTO	GT	IF
INITIALIZE	INP	INPUT	INT	INT%
LE	LEFT\$	LEN	LET	LINE
LOG	LPRINTER	LT	MATCH	MIDS
NE	NEXT	NOT	ON	OPEN
OR	OUT	PEEK	POKE	POS
PRINT	RANDOMIZE	READ	RECL	RECS
REM	REMARK	RENAME	RESTORE	RETURN
RIGHT\$	RND	SADD	SGN	SIN
SIZE	SQR	STEP	STOP	STR\$
SUB	TAB	TAN	THEN	TO
UCASE\$	USING	VAL	WEND	WHILE
	WIDTH	XOR		

### Expression Hierarchy

The Hierarchy for expression evaluation is as follows --

1. nested parentheses ( )
2. power operator ^
3. \* /
4. + - concatenation[+] unary[+ -]
5. relational operators:  
 < <= > >= = <> LT LE GT GE EQ NE
6. NOT
7. AND
8. OR XOR

### CBASIC: Predefined Functions

#### I/O Functions

- CONSTAT% - Returns the console status as an integer. If ready, a logical TRUE is returned.
- CONCHAR% - Reads one character from the console device.

#### Machine-Language Functions

- PEEK (<exp>) - Returns the contents of the memory location given by the expression.
- POKE <exp>, <exp> - Low-order eight bits of second expression are stored in memory location selected by first expression.
- CALL <exp> - CALL a machine language program at address specified.
- SAVEMEM <constant>, <exp>

- Reserve <constant> number of bytes and load the file specified by the string <exp> into the reserved area.

#### Numeric Functions

FRE	ABS(x)	INT(x)	INT%(x)
FLOAT(i%)	RND	SGN(x)	ATN(x)
COS(x)	EXP(x)	LOG(x) [e]	SIN(x)
SQR(x)	TAN(x)		

#### String Functions

ASC(a\$)	CHR\$(i%)	LEFT\$(a\$,i%)	LEN(a\$)
UCASE\$(a\$)	MATCH(a\$,b\$,i%)	MID\$(a\$,i%,j%)	RIGHT\$(a\$,i%)
STR\$(x)	VAL(a\$)	COMMANDS	SADD(a\$)

#### Disk Functions

RENAME(a\$,b\$) SIZE(a\$)

#### User-Defined Functions

The general forms are --

```
[<line number>] DEF <function name> [<dummy arg list>] = <expression>
```

and

```
[<line number>] DEF <function name> [<dummy arg list>]
.
.
.
[<line number>] FEND
```

#### 10.5.1 XREF.COM

The XREF.COM is a CBASIC utility for cross-referencing a CBASIC program. It produces a file which contains an alphabetized list of all identifiers used in a CBASIC program. The usage of the identifier (function, parameter, or global) is provided, as well as a list of each line in which that identifier is used. The file created has the same name as the CBASIC source file and is of type XRF. The standard output is 132 columns wide.

The following command is used to invoke XREF --

```
XREF <filename> [disk ref] [$<toggles>] ['title']
```

If the disk reference is specified, it instructs XREF as to what disk to place the output on.

The optional title field must be the last field in the command line. All characters following the first apostrophe on the command line up to the second apostrophe or until the end of the command line become the title. The title is truncated to 30 characters if the listing is 132 columns wide and 20 characters if the D toggle (80 column listing) is specified.

#### XREF Toggles

- Toggle A: Causes the listing to be output to the list device as well as the disk file.
- Toggle B: Suppresses output to the disk. If only the B toggle is specified, no output is produced.
- Toggle C: Suppresses output to the disk and permits output to the list device; same as A and B combined.
- Toggle D: Causes output to be 80 columns wide instead of 132.
- Toggle E: Produces output with only the identifiers and their usage.

For example, the following command produces a cross reference listing on the list device which is 80 columns wide --

```
A>XREF filename $CD
```

## 10.6 MBASIC

The MBASIC (Microsoft BASIC) Interpreter is invoked as follows --

```
MBASIC [<filename>][</F:<files>>][</M:<memory loc>>]
```

If <filename> is present, MBASIC proceeds as if a RUN <filename> command were typed after initialization is complete. A default extension of '.BAS' is assumed. If </F:<files> is present, it sets the number of disk data files that may be open at any one time during the execution of a program. The default here is 3. The </M:<memory loc> sets the highest memory locations that will be used by MBASIC. All memory to the start of FDOS is used by default.

In MBASIC, several Control keys have an effect.

^A	Enters Edit Mode on line being typed or last line typed
^C	Interrupts program execution and returns to MBASIC
^G	Rings <BELL> at terminal
^H	Deletes last char typed
^I	Tab (every 8)
^O	Halts/resumes program output
^R	Retypes the line currently being typed
^S	Suspends program execution
^Q	Resumes execution after ^S
^U,^X	Deletes line being typed
<CR>	Ends every line being typed in
<LF>	Breaks a logical line into physical lines
<DEL>	Deletes last char typed
<ESC>	Escapes Edit Mode Subcommands
.	Current line for EDIT, RENUM, DELETE, LIST, LLIST commands
&O,&	Prefix for Octal Constant
&H	Prefix for Hex Constant
:	Separates statements typed on the same line
?	Equivalent to PRINT statement

There are several commands that work when given at the command level.

Command	Syntax	Function
AUTO	AUTO [line][,inc]	Generate line numbers
CLEAR	CLEAR [,exp1][,exp2]]	Clear program variables; Exp1 sets end of memory and Exp2 sets amount of stack space
CONT	CONT	Continue program execution
DELETE	DELETE [[start][-[end]]]	Delete program lines
EDIT	EDIT line	Edit a program line
FILES	FILES [filename]	Directory
LIST	LIST [line[-[line]]]	List program line(s)
LLIST	LLIST [line[-[line]]]	List program line(s) on printer
LOAD	LOAD filename[,R]	Load program; ,R means RUN
MERGE	MERGE filename	Merge prog on disk with that in mem
NAME	NAME old AS new	Change the name of a disk file
NEW	NEW	Delete current prog and vars
NULL	NULL exp	Set num of <NULL>s after each line
RENUM	RENUM [[new][,old][,inc]]]	ReNUMBER program lines
RESET	RESET	Init CP/M; use after disk change
RUN	RUN [line number]	Run a prog (from a particular line)
	RUN filename[,R]	Run a prog on disk

SAVE	SAVE filename[,A or ,P]	Save prog onto disk; ,A saves prog in ASCII and ,P protects file
SYSTEM	SYSTEM	Return to CP/M
TROFF	TROFF	Turn trace off
TRON	TRON	Turn trace on
WIDTH	WIDTH [LPRINT] exp	Set term or printer carriage width; default is 80 (term) and 132 (prin)

Within EDIT mode, there are a number of sub-commands.

A	Abort -- restore original line and restart Edit
nCc	Change n characters
nD	Delete n characters
E	End edit and save changes; don't type rest of line
Hstr<ESC>	Delete rest of line and insert string
Istr<ESC>	Insert string at current pos
nKc	Kill all chars up to the nth occurrence of c
L	Print the rest of the line and go to the start of the line
Q	Quit edit and restore original line
nSc	Search for nth occurrence of c
Xstr<ESC>	Goto the end of the line and insert string
<DEL>	Backspace over chars; in insert mode, delete chars
<CR>	End edit and save changes

When declaring variables, there are several Variable Type Declaration Characters

\$	String	0 to 255 chars
%	Integer	-32768 to 32767
!	Single Precision	7.1 digit floating point
£	Double Precision	17.8 digit floating point

MBASIC recognises the following Program Statements (except I/O)

Statement Syntax	Function
CALL	CALL variable [(arg list)] Call assembly or FORTRAN routine
CHAIN	CHAIN [MERGE] filename [, [line exp] [, ALL] [, DELETE range]] Call a program and pass variables to it; MERGE with ASCII files allows overlays; start at line exp if given; ALL means all variables will be passed (otherwise COMMON only); DELETE allows deletion of an overlay before CHAIN is executed
COMMON	COMMON list of vars Pass vars to a CHAINED prog
DEF	DEF FNx[(arg list)]=exp Arith or String Function
	DEF USRn=address Define adr for nth assembly routine
	DEFINT range(s) of letters Define default var type INTEger
	DEFSNG " " " " " " " " Single
	DEFDBL " " " " " " " " Double
	DEFSTR " " " " " " " " String
DIM	DIM list of subscripted vars Allocate arrays
END	END

```

                Stop prog and close files
ERASE          ERASE var [,var ... ]
                Release space and var names
ERROR          ERROR code
                Generate error code/message
FOR            FOR var=exp TO exp [STEP exp]
                FOR loop
GOSUB          GOSUB line number
                Call BASIC subroutine
GOTO           GOTO line number
                Branch to specified line
IF/GOTO        IF exp GOTO line [ELSE stmt ... ]
                IF exp <> 0 then GOTO
IF/THEN        IF exp THEN stmt[:stmt] [ELSE stmt ... ]
                IF exp <> 0 then ... else ...
LET            [LET] var=exp
                Assignment
MID$           MID$(string,n[,m])=string2
                a portion of string with string2; start at pos n for m chars
NEXT           NEXT var[,var ... ]
                End FOR
ON ERROR       ON ERROR GOTO line
                Error trap subroutine
ON/GOSUB       ON exp GOSUB line[,line]
                Computed GOSUB
ON/GOTO        ON exp GOTO line[,line]
                Computed GOTO
OPTION         OPTION BASE n
                Min val for subscripts (n=0,1)
OUT            OUT port,byte
                Output byte to port
POKE           POKE address,byte
                Memory put
RANDOMIZE       RANDOMIZE [exp]
                Reseed random number generator
REM            REM any text
                Remark -- comment
RESTORE        RESTORE [line]
                Reset DATA pointer
RESUME         RESUME or RESUME 0
                Return from ON ERROR GOTO
                RESUME NEXT
                Return to stmt after error line
                RESUME line
                Return to specified line
RETURN         RETURN
                Return from subroutine
STOP           STOP
                Stop prog and print BREAK msg
WAIT           WAIT prot,mask[,select]
                Pause until input port [XOR select]
                AND mask <> 0

WHILE/         WHILE exp stmts ... WEND
WEND           Execute stmts as long as exp is true

```

When using PRINT USING statements, there are several 'Format Field

# Specifiers'

## a/ Numeric Specifiers

Specifier	Digits	Chars	Definition
£	1	1	Numeric field
.	0	1	Decimal point
+	0	1	Print leading or trailing sign
-	0	1	Trailing sign (- if neg, <sp> otherwise)
**	2	2	Leading asterisk
\$\$	1	2	Floating dollar sign; placed in front of leading digit
**\$	2	3	Asterisk fill and floating dollar sign
,	1	1	Use comma every three digits
^^^	0	4	Exponential format
_	0	1	Next character is literal

## b/ String Specifiers

Specifier	Definition
!	Single character
/<spaces>/	Character field; width=2+number of <spaces>
&	Variable length field

MBASIC has the following Input/Output Statements

CLOSE	CLOSE [[£]f[,£]f ... ] Close disk files; if no arg, close all
DATA	DATA constant list List data for READ statement
FIELD	FIELD [£]f,n AS string var [,n AS string var ...] Define fields in random file buffer
GET	GET [£]f[,record number] Read a record from a random disk file
INPUT	INPUT [;] [prompt string;] var [,var ...] INPUT [;] [prompt string,] var [,var ...] Read data from the terminal; leading semicolon suppresses echo of <CR>/<LF> and semicolon after prompt string causes question mark after prompt while comma after prompt suppresses question mark
KILL	KILL filename Delete a disk file
LINE INPUT	LINE INPUT [;] [prompt string;] string var Read an entire line from terminal; leading semicolon suppresses echo of <CR>/<LF>
LSET	LINE INPUT £f,string var Read an entire line from a disk file
LSET	LSET field var=string exp Store data in random file buffer left-justified or left-justify a non-disk string in a given field
OPEN	OPEN mode,[£] f,filename Open a disk file; mode must be one of -- I = sequential input file O = sequential output file R = random input/output file
PRINT	PRINT [USING format string;] exp [,exp ...] Print data at the terminal using the format specified

```

PRINT &f, [USING format string;] exp [,exp ...]
    Write data to a disk file
LPRINT [USING format string;] var [,var ...]
    Write data to a line printer
PUT      PUT [&] f [,record number]
    Write data from a random buffer to a data file
READ     READ var [,var ...]
    Read data from a DATA statement into the specified vars
RSET     RSET field var = string exp
    Store data in a random file buffer right justified or right
    justify a non-disk string in a given field
WRITE    WRITE [list of exps]
    Output data to the terminal
WRITE &f, list of exps
    Output data to a sequential file or a random field buffer

```

Mathematical expressions can have the following operators:-

```

=      Assignment or equality test
-      Negation or subtraction
+      Addition or string concatenation
*      Multiplication
/      Division (floating point result)
^      Exponentiation
\      Integer division (integer result)
MOD    Integer modulus (integer result)
NOT    One's complement (integer)
AND    Bitwise AND (integer)
OR     Bitwise OR (integer)
XOR    Bitwise exclusive OR (integer)
EQV    Bitwise equivalence (integer)
IMP    Bitwise implication (integer)
=,>,<, Relational tests (TRUE=-1, FALSE=0)
<=,<=,
>=,>=,
<>

```

The precedence of these operators is --

- |                               |                         |
|-------------------------------|-------------------------|
| 1. Expressions in parentheses | 8. Relational Operators |
| 2. Exponentiation             | 9. NOT                  |
| 3. Negation (Unary -)         | 10. AND                 |
| 4. *,/                        | 11. OR                  |
| 5. \                          | 12. XOR                 |
| 6. MOD                        | 13. IMP                 |
| 7. +,-                        | 14. EQV                 |

There are the following Arithmetic Functions

Function	Action
ABS(exp)	Absolute value of expression
ATN(exp)	Arctangent of expression (in radians)
CDBL(exp)	Convert the expression to a double precision number
CINT(exp)	Convert the expression to an integer
COS(exp)	Cosine of the expression (in radians)



CSNG(exp)	Convert the expression to a single precision number
EXP(exp)	Raises the constant E to the power of the expression
FIX(exp)	Returns truncated integer of expression
FRE(exp)	Gives memory free space not used by MBASIC
INT(exp)	Evaluates the expression for the largest integer
LOG(exp)	Gives the natural log of the expression
RND[(exp)]	Generates a random number exp <0 seeds new sequence exp =0 returns previous number exp >0 or omitted returns new random number
SGN(exp)	1 if exp >0 0 if exp =0 -1 if exp <0
SIN(exp)	Sine of the expression (in radians)
SQR(exp)	Square root of expression
TAN(exp)	Tangent of the expression (in radians)

There are the following String Functions

Function	Action
ASC(str)	Returns ASCII value of first char in string
CHR\$(exp)	Returns a 1-char string whose char has ASCII code of exp
FRE(str)	Returns remaining memory free space
HEX\$(exp)	Converts a number to a hexadecimal string
INPUT\$(length [,[\$f])	Returns a string of length chars read from console or from a disk file; characters are not echoed
INSTR([exp],str1,str2)	Returns the first position of the first occurrence of str2 in str1 starting at position exp
LEFT\$(str,len)	Returns leftmost length chars of the string expression
LEN(str)	Returns the length of a string
MID\$(string,start[,length])	Returns chars from the middle of the string starting at the position specified to the end of the string or for length characters
OCT\$(exp)	Converts an expression to an Octal string
RIGHT\$(str,len)	Returns rightmost length chars of the string expression
SPACES(exp)	Returns a string of exp spaces
STR\$(exp)	Converts a numeric expression to a string
STRING\$(length,str)	Returns a string length long containing the first char of the str
STRING\$(length,exp)	Returns a string length long containing chars with numeric value exp
VAL(str)	Converts the string representation of a number to its numeric value

There are the following I/O and Special Functions:-

Function	Action
CVI(str)	Converts a 2-char string to an integer
CVS(str)	Converts a 4-char string to a single precision number

CVD(str)	Converts an 8-char string to a double precision number
EOF(f)	Returns TRUE (-1) if file is positioned at its end
ERL	Error Line Number
ERR	Error Code Number
INP(port)	Inputs a byte from an input port
LOC(f)	Returns next record number to read or write (random file) or number of sectors read or written (sequential file)
LPOS(n)	Returns carriage position of line printer (n is dummy)
MKIS(value)	Converts an integer to a 2-char string
MKSS(value)	Converts a single precision values to a 4-char string
MKDS(value)	Converts a double precision value to an 8-char string
PEEK(exp)	Reads a byte from memory location specified by exp
POS(n)	Returns carriage position of terminal (n is dummy)
SPC(exp)	Used in PRINT statements to print spaces
TAB(exp)	Used in PRINT statements to tab to specified position
USR[n](arg)	Calls the user's machine language subroutine with the arg
VARPTR(var)	Returns address of var in memory or zero if var has not been assigned a value
VARPTR(&f)	Returns the address of the disk I/O buffer assigned to file number

MBASIC has the following Error Codes

1 NEXT without FOR	14 Out of string space
2 Syntax error	15 String too long
3 RETURN without GOSUB	16 String formula too complex
4 Out of data	17 Can't continue
5 Illegal function call	18 Undefined user function
6 Overflow	19 No RESUME
7 Out of memory	20 RESUME without error
8 Undefined line	21 Unprintable error
9 Subscript out of range	22 Missing operand
10 Redimensioned array	23 Line buffer overflow
11 Division by zero	26 FOR without NEXT
12 Illegal direct	29 WHILE without WEND
13 Type mismatch	30 WEND without WHILE

Disk Errors --

50 Field overflow	58 File already exists
51 Internal error	61 Disk full
52 Bad file number	62 Input past end
53 File not found	63 Bad record number
54 Bad file mode	64 Bad file name
55 File already open	66 Direct statement in file
57 Disk I/O error	67 Too many files

## 10.7 BASCOM

BASCOM is the compiler version of MBASIC. The following direct mode commands are NOT implemented on the compiler and will generate an error message --

AUTO	CLEAR	CLOAD
CSAVE	CONT	DELETE
EDIT	LIST	LLIST
RENUM	COMMON	SAVE
LOAD	MERGE	NEW
ERASE		

The following statements are used differently with the compiler than with the interpreter (refer to the manual for details) --

CALL	DEFINT	DEFSNG
DEFDBL	DEFSTR	DIM
ERASE	END	ON ERROR GOTO
RESUME	STOP	TRON
TROFF	USRn	

The compiler is invoked by the BASCOM command; it may be called by --

A>BASCOM

or

A>BASCOM command line

where "command line" is --

[dev:][obj file],[dev:][lst file]=[dev:]source file[/switch ...]  
 where 'obj file' is the name of the object (.REL) file, 'lst file' is the name of the listing file (.PRN) and 'source file' is the name of the source BASIC file. The '.PRN' file can be sent to the LST: device.

If just BASCOM is used, the user will be prompted with an asterisk, after which he should enter the command line.

Switches --

/E Use this switch if ON ERROR GOTO with RESUME <line number> is used  
 /X Use this switch if ON ERROR GOTO with RESUME, RESUME 0, or RESUME NEXT is used  
 /N Do not list generated object code  
 /D Generate debug/checking code at runtime  
 /S Write quoted strings of more than 4 chars as they are encountered  
 /4 Recognize Microsoft 4.51 BASIC Interpreter conventions  
 /C Relax line numbering constraints; lines need not be numbered sequentially; /4 and /C may not be used together  
 /Z Use Z80 opcodes

BASIC Compiler Error Messages

1/ Compile-Time Fatal Errors

SN	Syntax error	OM	Out of memory
SQ	Sequence error	TM	Type mismatch
TC	Too complex	BS	Bad subscript
LL	Line too long	UC	Unrecognizable command

OV	Math overflow	/O	Division by zero
DD	Array already dim'ed	FN	FOR/NEXT error
FD	Function already def	UF	Function not defined
WE	WHILE/WEND error	/E	Missing /E switch
		/X	Missing /X switch

## 2/ Compile-Time Warning Errors

ND	Array not dimensioned	SI	Statement ignored
----	-----------------------	----	-------------------

## 3/ Run-Time Error Messages

2	Syntax error	52	Bad file number
3	RETURN without GOSUB	53	File not found
4	Out of data	54	Bad file mode
5	Illegal function call	55	File already open
6	Floating/Integer ovfl	57	Disk I/O error
9	Subscript out of range	58	File already exists
11	Division by zero	61	Disk full
14	Out of string space	62	Input past end
20	RESUME without error	63	Bad record number
21	Unprintable error	64	Bad filename
50	Field overflow	67	Too many files
51	Internal error		

## 10.8 PASCAL MT+

The Pascal/MT compiler exists in two versions, each consisting of two 8080 object code files: FLTCOMP.COM and P2/FLT.OVL for the version in which REAL numbers are implemented as floating point values internally and BCDCOMP.COM and P2/BCD.OVL for the version in which REAL numbers are implemented as BCD values internally. These files are Pass 1 and Pass 2 of the Pascal/MT compiler, respectively. Also required by the compiler are the following files --

P1ERRORS.TXT	- Pass 1 Error Messages
P2ERRORS.TXT	- Pass 2 Error Messages
PASCAL/F.RTP	- Run-time Package (including debugger) for FLT
PASCAL/B.RTP	- Run-time Package (including debugger) for BCD

The input files to the Pascal/MT compiler must have the extension '.SRC' or '.PAS' indicating that it is a source program file. There must be a carriage return/line feed sequence at the end of each input line and an input line may not be longer than 80 characters.

The Pascal/MT compiler is invoked by using the following command --

PASCAL filename.DL

where 'filename' is the name of the file with the extension '.SRC' or '.PAS' containing the Pascal/MT source statements to be compiled, D is Y or N to indicate whether to include the debugger in the resultant '.COM' file, and L is Y or N to indicate whether to produce a '.PRN' file (listing). PASCAL defaults to no debugger and no listing. The four invocation options are --

```
PASCAL filename      - no debugger, no listing
PASCAL filename.Y    - debugger, no listing
PASCAL filename.NY   - no debugger, listing
PASCAL filename.YY   - debugger, listing
```

For usage with the BCD version of the compiler, the commands are similar except that 'PASCAL' is replaced by 'BCDCOMP'.

#### Compilation Switches

Compile-time options may be specified to the compiler from within the source file. Such options take the form of special comments. The form of these comments is

(\*So info\*) or (\$o info)

where 'o' is the letter of the option and 'info' is information particular to that option. These options are --

```
$I<filename>  Include <filename>.SRC into source stream
$L+ or $L-    Turn listing on (default) or Turn listing off
$P           Insert form feed into '.PRN' file
$D+ or $D-    Turn debug code on (default) or Turn debug code off
$C+          Use CALL instructions for real operations
$Cn          Use RST n for real operations (n=0 ... 7)
$O $aaaa     ORG program (run-time) at $aaaa (default 100H)
$R $bbbb     ORG RAM data at $bbbb
$Z $cc00     Set run-time size to $cc 256-byte pages
$X $dddd     Set run-time stack space to $dddd (default $200)
$S+ or $S-    Turn recursion on or Turn recursion off (default)
$Q+ or $Q-    Enable verbose output (default) or Disable verbose
```

#### File Input/Output

The standard Pascal READ, READLN, WRITE, and WRITELN statements are implemented for the CP/M console device. WRITE or WRITELN to a built-in file called PRINTER is allowed to directly access the CP/M list device (like, WRITE(PRINTER,'Hello')).

The following extensions are implemented to handle files --

```
OPEN(fcbname,title,result(,extent_number));
    (extent_number defaults to 0)
CLOSE(fcbname,result);
CREATE(fcbname,title,result);
DELETE(fcbname);
BLOCKREAD(fcbname,buffer,result(,relativeblock));
BLOCKWRITE(fcbname,buffer,result(,relativeblock));
```

where fcbname : a variable of type TEXT (array 0..32 of CHAR)  
 title : ARRAY [0..11] of CHAR with  
           title[0]=disk select byte (0=logged in disk,

```

        1=A,...)
        title[1..8]=filename and title[9..11]=filetype
result   : integer to contain returned value
buffer   : ARRAY [0..127] of CHAR
relativeblock : optional integer 0..255

```

## Special Functions and Procedures

Pascal/MT supports the following special routines --

```

PROC MOVE(source,dest,length-in-bytes);
PROC EXIT;
FUNC TSTBIT(16-bit-var,bit&):BOOLEAN;
PROC SETBIT(VAR 16-bit-var,bit&);
PROC CLRBIT(VAR 16-bit-var,bit&);
FUNC SHR(16-bit-var,&bits):16-bit-result; (Shift Right)
FUNC SHL(16-bit-var,&bits):16-bit-result; (Shift Left)
FUNC LO(16-bit-var):16-bit-result;
FUNC HI(16-bit-var):16-bit-result;
FUNC SWAP(16-bit-var):16-bit-result;
FUNC ADDR(variable reference):16-bit result;
PROC WAIT(portnum:constant; mask:constant; polarity:boolean);
FUNC SIZEOF(variable or type name):integer;

```

Please refer to pp 32-33 of "Pascal/MT 3.0 Guide" for further info.

### 10.8.1 Pascal/MT Symbolic Debugger

The debugging facilities available to the user when using the debugger fall into two categories -- program flow control and variable display.

If the user wishes to see the commands during the execution of the debugger, type a '?' followed by a return.

The program flow commands provided in the symbolic debugger allow the user to debug the Pascal/MT program at the Pascal source statement level. Included are go/continue (with optional breakpoint), trace, set/clear/display permanent breakpoint and a mode which will display the name of each procedure/function on the console as the procedure or function is entered.

These commands are discussed briefly on the following displays --

Debugger Command: G - Go with optional breakpoint

```

Syntax:  G(<linenumber>)
         G(<proc/func name>)

```

Go resumes execution where the program last stopped. Breakpoint may be optionally set at a specific line or function/procedure.

Debugger Command: T - Trace

```

Syntax:  T(<integer>)

```

Execute one or more lines of the program.

Debugger Command: E - Procedure/Function Display Toggle

Syntax: (-)E

E engages display of the names of procedures/functions entered; -E disengages it.

Debugger Command: S - Set/Clear Slow Execution Mode

Syntax: (-)S

S allows the user to select Fast, Medium, or Slow execution speed; -S causes the program to run at full speed.

Debugger Command: P - Set/Clear Permanent Breakpoint

Syntax: -P (\* Clears breakpoint \*)  
P<linenumber>  
P<proc/func name>

P sets the permanent breakpoint; -P clears it.

Debugger Command: B - Display Permanent Breakpoint

Syntax: B

Displays line the permanent breakpoint is set for.

Debugger Command: D - Variable Display

Syntax: D <global var>  
D <proc/func name>:<local var>  
D <func name>  
D <pointer name>^

The D command is used as indicated.

Debugger Commands: +,-,\* - Variable Display

Syntax: \* -- display last value requested (using D or some other)  
+n -- display variable n bytes forward from last  
-n -- display variable n bytes backward from last

Pascal/MT Reserved Words

ABS	DO	LO	READ	TSTBIT
ADDR	DOWNT0	MAXINT	READLN	TYPE
AND	ELSE	MOD	REAL	UNTIL
ARRAY	ENABLE	MOVE	RECORD	VAR
BEGIN	END	NIL	REPEAT	WAIT
BLOCKREAD	EXIT	NOT	RIM85	WHILE
BLOCKWRITE	EXTERNAL	ODD	ROUND	WRITE
BOOLEAN	FALSE	OF	SETBIT	WRITELN
CASE	FILE	OPEN	SHL	
CHAIN	FOR	OR	SHR	
CHAR	FUNCTION	ORD	SIM85	
CHR	GOTO	OUTPUT	SIZEOF	
CLOSE	HI	PACKED	SQR	
CLRBIT	IF	PRED	SQRT	
CONST	INLINE	PRINTER	SUCC	
CREATE	INPUT	PROCEDURE	SWAP	

DELETE	INTEGER	PROGRAM	THEN
DISABLE	INTERRUPT	RANDOMREAD	TO
DIV	LABEL	RANDOMWRITE	TRUE

## Notes

1. Hexadecimal values may be specified as \$hhhh, like \$1A = 1AH.
2. All standard Pascal type definitions except ARRAY are supported. The standard form ARRAY...OF ARRAY... must be specified as ARRAY[.....], and a maximum of three dimensions may be used.
3. Type TEXT is ARRAY [0..35] OF CHAR.
4. Interrupt Procedures, declared as "PROCEDURE INTERRUPT[i] proc;", are supported, where i is the restart vector number (0..7).
5. CP/M V2 random file access is supported by RANDOMREAD and RANDOMWRITE.
6. Machine code, constant data, and assembly language code may be inserted using INLINE (see pp 37-39 of "Pascal/MT 3.0 User Guide").
7. Chaining is supported by CHAIN, whose usage is "CHAIN(filename)".
8. Redirected I/O is supported (see pp 42-43).



## CHAPTER 11 – RMAC, Using a Relocating Macroassembler

On the CP/M distribution disc, there exists an good relocating Macroassembler; RMAC. If you have already cut your teeth on the 8080 assembler ASM, then you will find that RMAC and its attendant utilities will provide some excellent new facilities. RMAC is essentially an upgrade of ASM. Once you have learned some of RMACs eccentricities and bugs, it is a useful tool. As well as the ability to understand Z80 opcodes (admittedly of a non-standard version) a macro assembler allows you to build up your own mini-language.

No programmer likes to do the same thing twice. A programmer's intrinsic laziness is one of his better assets. A relocating macroassembler allows you to build up a library of commonly used functions, procedures and subroutines that can be used from an assembly program or even from high-level languages. The Macro facilities then allow you to construct your own time-saving mini-language, or, at least, avoid unnecessary repetition in your code.

Let's try out a simple example. No one is going to pretend that this is a particularly useful example, but it serves to illustrate the virtues of relocatable code and macros:-

The classic 'first program' is to display on the screen the message "Hello world!".

Under CP/M, this is very simple. We just do the following, calling it HELLO.ASM.

```
;  
;the macro-less version  
  
      bdos      equ      0005H      ;address of the BDOS entry point  
  
start: mvi      c,9      ;bdos code for printstring  
       lxi      d,hello$string ;address of the friendly string  
       call     bdos      ;bang the button  
       ret      ;and return to the CCP  
  
hello$string: db      'Hello World!$'  
  
end start      ;Begin execution at the label START  
;  
;
```

One can produce a 'COM' file of this by the following commands:-

```
RMAC HELLO  
LINK HELLO
```

There are a few things to notice about this. Firstly, it is not a very useful program; secondly, it contains no 'ORG' instruction to place the code at a particular location (this is all looked after by the linker LINK), and thirdly, we have to tell the linker where the program starts by putting a label after the 'END' statement. This tells the LINK program to vector program execution to the line where 'start' is defined.

We can simplify the operation, particularly if we are going to write more than one string during the execution of the program.

```

bdos    equ    0005H           ;address of the BDOS entry point

;---Macro Definition---;
print macro    string
    local    where

        mvi    c,9           ;bdos code for printstring
        lxi    d,where       ;address of the friendly string
        call   bdos          ;bang the button
        dseg

where:    db    string
        db    '$'           ;insert the CP/M terminating $ sign
        cseg
    endm

;---Program Start---;

start:   print    'Hello World'

        ret

end start

```

We have, here, introduced some important new features of the assembler. (the Intel and Microsoft assembler will assemble all this correctly too). We have declared a macro which, at assembly time, substitutes the text up to the 'endm' statement. The macro is fairly difficult to comprehend if you are unfamiliar with such things, but, once constructed, can be used for ever and ever without any more thought. In its simplest form, it allows a whole chunk of text to be represented by one word, the typical 'lazy programmer' device. Let us look at the macro definition in more detail:-

We are not going to attempt an exhaustive explanation of the Intel Macro language statements here, we are more concerned with trying to explain the purpose of them. The line "print macro string" says that the following lines up to the matching "endm" are the body of a macro definition called 'print' that has one parameter. For the macro, we need some way of producing a unique label every time the macro is invoked. The 'local' statement in the next line "local where" tells the assembler to substitute a unique label for 'where' every time the macro is invoked.

The next three lines are the assembler ops as in the first simple example. We then meet the "DSEG" statement. This requests the assembler to place the following assembler statements into the 'data segment'. This will be located after the program code. we do not have to worry where about this will go as the linker will sort all this out for us. The result is that we put the string safely out of the way. (if we left out the DSEG statement, the program would try to execute the data with nasty results.) With the string and its terminator safely defined, we return to the code segment with the "CSEG" statement.

The time has come to start to build a library. We need a library of all our favourite macros which we will call MYMACRO.LIB. If we place our macro definition along with the BDOS equate into our new library, our program looks rather slimmer.

```
maclib  MMACRO      ;read in our macro library
```

```
;--Program Start--;
```

```
start: print  'Hello World'
       ret
```

```
end start
```

This still is not very useful, is it? We really need a print statement which is rather more versatile. What about extending it to mimic the complex 'printf' statement of the language C? This is not too hard, and once done is of permanent use. Let us keep it simple for the time being, just allowing the insertion of decimal and hexadecimal values as well as other strings. Let us also set up a mechanism whereby we can send simple control codes out to the screen such as carriage returns and line feeds. To give it all the other 'C style' features is just a matter of increasing the complexity of the code rather than the underlying algorithms and would be good practice for you anyway.

What we want is something that would work the following program, which converts an number into its HEX equivalent:-

```
maclib  MMACRO      ;read in the macro library
```

```
;--Input Buffer--;
```

```
InputBuffer: ds      2      ;the input buffer for buffered keyboard input
OurString    ds      9      ;the string taken from the keyboard
```

```
value:       dw      0
```

```
;--Program Start--;
```

```
error:
```

```
    printf  '\n Sorry, but %s was not a valid number \n',OurString
```

```
start:
```

```
    orError error      ;if there is an error, jump to the label 'error'
    input  'Give me a number ->',InputBuffer,8
    convert OurString,value ;convert the string into a binary integer
    printf '\n\\The value of %s is %d in hexadecimal',OurString,value
    ret                      ;finish the program
```

```
end start
```

This does not look much like assembly language but it is! All we need to do is to write a support library and macro library to make it happen and we have suddenly got a personal programming language. This is rather going to extremes but should illustrate the usefulness of the macro facility.

The input statement is the simplest to do.

```
;--INPUT Macro Definition--;
```

```
input  macro prompt,buffer,max
```

```
;;reads an input line from the keyboard into a buffer, allowing a maximum of
;; 'max' characters. If the prompt is omitted, then none is displayed.
```

```

;; If the buffer address is omitted it defaults to the default CP/M buffer
;; at 0080H and if max is omitted, it defaults to 80 characters!

;;usage examples:-

;;buffer:      ds      82

;;      input  'What is your name? ->',buffer,10
;;this prompts with the string and put the results in 'buffer' up to 10 chars
;;      input  'What is your name?'
;;this prompts with the string and put the results at 0080H up to 80 chars
;;      input
;;this inputs a string and put it at 0080H up to 80 chars
;;      input  '',10
;;this inputs a string and put it at 0080H up to 10 chars
;;      input  'What is your name? ->',,,10
;;this prompts with the string and put the results at 0080H up to 10 chars

      local  where

      if not nul prompt

      mvi    c,9           ;bdos code for printstring
      lxi    d,where       ;address of the friendly string
      call   bdos          ;bang the button

      dseg
where: db    prompt
      db    'S'           ;insert the CP/M terminating $ sign
      cseg
      endif
      if nul max           ;if he did not specify any maximum value to
                           ;be typed in from the keyboard
      maximum set 80
      else
                           ;if he specified a value for the max to be
                           ;typed in from the keyboard
      maximum set max
      endif

      if nul buffer       ;if he did not specify where he wanted the
                           ;string
      lxi    d,0080H
      else
                           ;he did specify where he wanted the string put
      lxi    d,buffer
      endif
      mvi    a,maximum
      stax   d
      mvi    c,0aH        ;the CP/M Read Console Buffer statement
      call   bdos         ;push the button

      endm

```

There is nothing complicated about this, but we may want to extend the prompting facilities later on to include the acceptance of control codes and so on. Note that it allows for various form of macro invocation, depending on the

purposes required and the laziness of the programmer. It is always worth putting in sensible defaults using the useful 'NUL' statement which merely tests if anything was entered for the particular parameter. It is interesting to assemble various examples of this macro to see how compact the resulting code actually is.

For the other statements, we will need to start a support library. If you have ever used a language compiler, you will have noticed that they all have support libraries that are searched at linkage time for unresolved symbols. It pulls in all the subroutines that are referenced anywhere in the program, and therefore means that you do not need to include all the code for the supporting routines in your program source module. This speeds up program development by a huge factor.

You will notice that we are going to include error trapping. Our support library will jump to an error routine if an error occurs. With the macro `on$Error`, one can change the location to which the error is vectored. In the course of a program of any length, you will want to divert the error handler in all sorts of ways. Here, we merely want it to repeat the input request. The functions and subroutines will jump to this routine if an error is found.

We also need a routine to convert a string into a binary integer, and a routine to imitate the Unix C `printf` function. Here are the various routines that would need to go into the library. They seem rather complex for the present program, but a realistic program would need to exercise them rather more. Normally, the I/O and formatting routines would be in separate modules within a library so that the whole lot would not be pulled into the program when only a small portion was needed. Once assembled, these routines can be forgotten about.

```

;-----
NAME 'IO'
;-----
;--Program Entry Points--
public designed
public Default$ErrorHandler ;where errors go until told otherwise
public tyo ;puts the character in A out to the screen
public Strout ;sends null-terminated string at HL to the screen
public do$printf ;see the routine itself!
public Format ;see the routine itself
public INTEGER ;convert a number in memory from ASCII to Binary.
;HL points to the null-terminated string. Returns
;with value in HL and A.
public error$vector ;the pointer to the current error handler

;--Program Addresses --;
BIOS Equ 0000H ;The jump vector to the warm boot routine
BDOS Equ 0005H ;the jump vector to the Bdos

;--Data--;

DSEG ;Put the following in the data segment
buffer$base: ;our working buffer
ds 160 ;workspace

error$vector dw default$ErrorHandler ; vector to the error handler

err$message

```

```

                db      'Error, please try again'      ;default error message
CSBG           ;back to the code segment

;-----
Default$Error$Handler: ;just say there has been an error and leave the
;program

                lxi     d,err$message      ;address of error string in DE
                mvi     c,9                ;and the print string code in C
                call    bdos               ;bang the button
                jmp     0000H              ;and leave the program

;-----
Go$Error:       ;execute whatever error routine is in operation
                pop     h                  ;go down a stack level
                lhd     Error$Vector       ;Get the current error handler
                pchl                    ;and jump there

;-----
;console output routines
;-----
tyo:            ;send the character in the A register out to the screen
                mov     c,a                ;put it in the C register for the BIOS
                lhd     BIOS+1             ;Pick up address of the BIOS jump vector table
                lxi     d,9                ;index into it to the CONOUT routine
                dad     d
                pchl                    ;and jump to the Conout routine

;-----
;-----
$ROUT:          ;sends out null terminated string pointed-to by HL to the
;CON: device using the direct BIOS route
                mov     a,m                ;are we at the NULL terminator
                ora     a                  ;set flags
                rz                     ;return if we hit the null terminator as the string is done
;
;
                push    h                  ;save the pointer into the string
                call    tyo                ;and place current character on the console
                pop     h                  ;send out null-terminated string
                inc     h                  ;and then bump the pointer
                jmp     strout              ;to console

;-----
; String Formatting routines
;-----
do$printf:      ;print a string pointed to by HL, using parameters pointed
;to by pointer array addressed by DE

                lxi     b,buffer$base     ;use the default workspace
                call    format             ;and format the string as required
                lxi     h,buffer$base     ;point to the formatted string
                call    strout             ;and blast it out
                ret

```

```

;-----
format:      ;format string pointed to by HL into string area pointed
;to by BC, using parameters pointed to by pointer array addressed by DE

??do$another:
    mov     a,m      ;get the next byte from the formatting model
    cpi     '%'       ;is it a parameter?
    jz      do$parameter ;if so, then there is work to be done
    cpi     '\''      ;can it be a control character?
    cz      do$control  ;if so we need to interpret this
    stax    b         ;store it
    ana     a         ;is it the end of the string?
    rz      ;return if so
    inc     h         ;otherwise bump the pointer into the string
    inc     b         ;and the pointer to the formatted string
    jmp     ??do$another ;and do the next character

do$control:   ;substitute the control character
    inc     h         ;get next character
    mov     a,m       ;what is it?
    cpi     't'       ;Hmm.. could it be a tab that he wants?
    jnz     ??not$tab ;jump if not
    mvi     a,09H     ;this is simple, we just put a tab in the
    ret      ;formatted dstring and return with the tab substituted for the t
??not$tab:   ;it was not a t
    cpi     'b'       ;
    jnz     ??not$b   ;
    mvi     a,08H     ;it's a backspace
    ret

??not$b:
    cpi     'r'       ;
    jnz     ??not$r   ;
    mvi     a,0DH     ;it is a carriage-return (without line feed)
    ret

??not$r:
    cpi     'f'       ;
    jnz     ??not$f   ;
    mvi     a,0CH     ;it is a form-feed
    ret

??not$f:
    cpi     'n'       ;does he want a line-feed carriage-return?
    rnz     ;if it was none of these, we are baffled
    mvi     a,0DH     ;put the carriage return in the formatted string
    stax    b         ;
    inc     b         ;bump the pointer into the formatted string
    mvi     a,0AH     ;and put in a line feed
    ret      ;then go home

do$parameter: ;do the specified parameter
    inc     h         ;roll up our sleeves, and bump the pointer
    mov     a,m       ;get the type of the object (variable)
    ana     a         ;has the programmer gone potty?
    rz      ;silly string, must be all done
    push    h         ;as we are not using this for a while
    push    psw       ;save the formatting type
    ldax    d         ;get what is pointed at by DE
    mov     l,a       ;into the HL pair

```

```

    inx    d        ;get the high byte
    ldax   d
    mov    h,a      ;we have a pointer to the parameter
    inx    d        ;DE is bumped to the next parameter
    pop    psw      ;restore the formatting type
    cpi    's'      ;is it a string?
    jnz    ??not$a$string ;jump if it is not a string
;This puts the string pointed to by HL onto the string pointed to by DE
??same$a$again:      ;send out the string until we get the null terminator
    mov    a,m      ;is it the null-terminator
    ana    a        ;if so job done
    jz     ??job$done ;time to go back to format the next byte
    stax   b        ;store it into the string.
    inx    h        ;bump the pointer into the string variable he specified
    inx    b        ;and the pointer into the formatted string
    jmp    ??same$a$again ;and do the next byte

??not$a$string: ;so it was not a string.
    cpi    'u'      ;is it a decimal value
    jnz    ??not$unsigned ;if not, then jump
    call   dodec     ;append decimal representation of value pointed
;to by HL onto the string at BC
    jmp    ??job$done ;and go back to formatting the next byte

??not$unsigned: ;so what was it?
    cpi    'd'      ;is it a signed decimal integer
    jnz    ??not$decimal
    call   dosigned
    jmp    ??job$done

??not$decimal:
    cpi    'c'      ;is it a single character
    jnz    ??not$character
    mov    a,m      ;then get it
    stax   b
    inx    b
    jmp    ??job$done

??not$character:
    cpi    'x'      ;is it hex?
    jnz    ??default ;we are about to give up
    call   dohex     ;append hex representation of value pointed
;to by HL onto the string at BC
    jmp    ??job$done ;and go back to do the next bit of formatting
??default: ;if it was unrecognized by the formatter
    push   psw      ;we should pass it across to the formatted string
    mvi    a,'%'    ;put in the % sign
    stax   b        ;and.
    inx    b        ;.then
    pop    psw      ;and the formatting byte
    stax   b
??job$done: ;the job has been done
    pop    h        ;restore the pointer to the original string
    inx    h        ;bump the pointer into the formatting string
    jmp    ??do$another ;and go do another

```

---

```

;      Support routines

```



```

;
HexOut: ;append ASCII hex representation of byte value pointed
;to by HL into the two bytes pointed-to by BC, bumping BC

        mov     a,m           ;get the byte to interpret
        rrc
        rrc
        rrc
        call    outchr        ;do the high nibble first, please
        mov     a,m           ;get the low nibble
outchr: ;interpret the low nibble of A
        ani     0FH           ;mask out the high byte
        adi     90H           ;do the intel trick
        daa
        aci     40H           ;baby's first word
        daa
        stax    b             ;store it in *BC
        inc     b             ;and bump the pointer
        ret                ;return with the job done

;
DOHEX:   ;append hex representation of word value pointed
;to by HL onto the string at BC, Bumping BC.
        inc     h             ;do the high byte first
        call    hexout        ;convert the byte
        dcx     h             ;now do the low byte
        call    hexout        ;converting it
        ret

;
dosigned: ;append the signed decimal representation of value pointed to by HL
;onto the string at BC
        xra     a
        jmp     to$dec

;
dodec:   ;append the unsigned decimal representation of value pointed to by HL
;onto the string at BC
        xra     a
        dcr     a             ;put non-zero in A to signify unsigned
to$dec:
        push    h             ;what is a stack for anyway
        push    d             ;save HL and DE
        mov     e,m           ;get what is pointed-to by HL into DE
        inc     h
        mov     d,m           ;get the value itself
        ana     a             ;is it unsigned
        jnz     ??Unsigned
        mov     a,d
        ana     a             ;is the number positive
        jp      ??Unsigned
        cma
        ;if negative, then negate it
        mov     d,a
        mov     a,e           ;DE=DE
        cma
        mov     e,a

```

```

    inx     d
    mvi     a,'-'      ;and now put in a minus sign
    stax    b           ;in the formatted string
    inx     b
??Unsigned:
    mov     l,c
    mov     h,b         ;save its destination
    shld    to$where     ;and remember it in a variable
    xchg
    call    decout       ;do the recursive routine
    lhld    to$where
    mov     c,l
    mov     b,h         ;restore the pointer
    pop     d           ;restore HL and DE
    pop     h
    ret              ;and go home

;-----
DECOUT: ;puts the decimal representation of the value in HL on the
;string pointed to by to$where

    push    b
    push    d
    push    h           ;save primary registers
    lxi     b,-10       ;-Radix
    lxi     d,-1        ;prime the DE pair
??1xloop:
    dad     b           ;repeated subtraction
    inx     d
    jc      ??1xloop
    lxi     b,10
    dad     b           ;add in the radix
    xchg
    mov     a,h
    ora     l
    cnz     decout      ;recursive call
    mov     a,e
    adi     '0'         ;convert binary into Decimal
    lhld    to$where     ;store the byte
    mov     m,a         ;by picking up the pointer, storing the byte....
    inx     h           ;..and bumping the pointer
    shld    to$where     ;and bump the pointer
    pop     h           ;restore the primary registers
    pop     d
    pop     b
    ret

to$where: dw 00        ;local variable for recursive routine

;-----
ConUC:   ;convert a character into upper case if it is lower case
    cpi    7FH
    mc
    cpi    61H
    rc
    ani    5FH
    ret

```

---

```

;
IsIt$Terminator:    ;is the character in A valid as a number in any radix
;returns non-zero if invalid, else zero.

```

```

        cpi    '0'    ;is it a number?
        jc     ??no
        cpi    '9'+1
        jc     ??yes
        call   conuc   ;convert to upper case
        cpi    'A'
        jc     ??no
        cpi    'F'+1
        jc     ??yes
        cpi    'H'    ;is it a hex specifier?
        rz
        cpi    'O'    ;is it an octal specifier?
        rz
??no:   ana    a        ;set non-zero
        ret
??yes:  cmp    a        ;set zero
        ret

```

```

INTEGER:          ;convert a number in memory from ASCII to Binary. HL points
;to the null-terminated string. Returns with value in HL and A.
;Either decimal, hexadecimal or octal values will be accepted and may be
;signed. On exit, DE points to the character after the end of the string.

```

```

        mvi    a,10    ;assume that is is a decimal number..
        sta    ??Radix ;default radix
??Once$More:
        mov    a,m      ;index over leading spaces
        cpi    ' '
        jnz    ??Hit$String
        inc    h
        jmp    ??Once$More

??Hit$String:
        mov    a,m
        cpi    '+'
        jz     ??plus
        sui    '-'    ;is it a minus sign?
        sta    ??negate;remember the sign
        jnz    ??done$sign

??plus:
        inc    h        ;increment over the minus sign
??done$sign:
        push   h        ;save pointer
;has it got a radix qualifier?
??back0:    ;lets find the end of the string
        mov    a,m      ;by going through til we find the NULL
        ana    a        ;have we reached the null terminator?
        jz     ??got$to$end    ;if so, then that is good
        call   isIt$Terminator
        jnz    ??got$to$end
        inc    h        ;otherwise try the next byte
        jmp    ??back0

```

```

??got$to$end: ;we have found the end of the string
               dcx h ;get the last character
               mov a,m ;into a.
               cpi '9'+1 ;is it a letter?
               jc ??no$radix ;if not a letter, then it was a decimal number
               mvi m,0 ; substitute a null terminator
               lxi h,??Radix ;prepare to write to RADIX
               call conuc ;convert to upper case if necessary
               cpi 'H' ;was it a Hexadecimal specification?
               mvi m,16
               jz ??1on ;if so, jump
               cpi 'O' ;was it octal?
               mvi m,8
               jz ??1on ;if so then jump
               cpi 'B' ;was it binary
               mvi m,2
               jz ??1on
               mvi m,10 ;if none of these, it was decimal

??1on:
??no$radix:
               pop h ;restore our pointer
               lxi d,0 ;initialize accumulator
               xchg
??2loop:
               ldax d ;get the next digit
               ana a ;was it the end of the string
               jz ??All$Done ;all done
               call IsIt$S$Terminator
               jnz ??All$Done
               sui '0' ;make character binary
               cpi 10 ;was it a hex digit?
               jc ??hoppity ;if not, jump
;adjust A..F to binary
               sui ('A'-'9')-1 ;convert it
??hoppity:
               push h ;save our total
               lxi h,??Radix ;what radix is this
               cmp m
               cmc ;is it illegal in terms of this radix?
               pop h ;restore the total
               jc Go$Error ;illegal character
               inc d ;increment pointer
               mov c,a ;put the value represented by the digit in BC
               mvi b,0
               lda ??Radix ;get the radix
               dad h ;double existing total
               cpi 2 ;if binary radix, enough done
               jz ??onwards
               push b ;save value represented by digit
               push h ;save our 16-bit accumulator
               dad h ;*4
               dad h ;*8
               pop b ;restore the binary value of the current digit
               cpi 8 ;was it an octal radix?
               jz ??over$we$go ;if so, no problem
               cpi 16 ;was it hex?
               jz ??over$we$go
               dad b ;add in 16-bit Acc*2 to get decimal

```

```

??over$we$go:
    pop    b            ;restore the value represented by the digit
    cpi    16           ;is it a HEX value
    jnz    ??onwards
    dad    h            ;if HEX, then double the 16-bit accumulator
??onwards:
    dad    b            ;add in the new value
    jmp    ??2xloop     ;do the next
??All$Done:
    lda    ??Negate
    ana    a            ;must we negate the result?
    mov    a,l
    rnz    ;return if not
    cma    ;so let us negate the result
    mov    l,a
    mov    a,h
    cma
    mov    h,a          ;HL←HL
    inx    h
    mov    a,l
    ret

??Radix:    DB    10            ;this is where we store the radix
??Negate:   DB    0FFH         ;zero if we negate the result

```

---

Once these are written, we can forget the details except on how we use them. If we use only macros in our main files, then we can even transport all our code from one processor to another merely by rewriting the macro and support libraries. We are now ready for the other macros:-

```
bdos    equ    0005H
```

;-PRINTF Macro Definition-;

```

printf    macro    string,am,bm,cm,dm,em,fm,gm
    local    x,y            ;;we have two local variables

    extrn    dc$printf      ;;this is in the support library
    push    b              ;;save all the primary registers
    push    d
    push    h
    times   set    0
    DSEG                      ;;now construct a table for as many params
                                ;;as were specified

x:
    irp     z,<am,bm,cm,dm,em,fm,gm>
    times   set    times+1
    if     nul    z
    exitm
    endif
    dw     z            ;parameter times
    endm

    CSEG
    lxi    d,x          ;;put address of parameter table in DE

```

```

        DSEG                ;;put the string in the data segment
y:      db      string,0
        CSEG
        lxi     h,y          ;;and address of formatting string in HL
        call   do$printf     ;;print it
        pop    h
        pop    d
        pop    b
        endm

```

;-ON\$ERROR Macro Definition-;

```

on$Error      macro          whereto
        extrn   error$vector ;;it is in our support library
        lxi     h,whereto
        shld    error$vector
        endm

```

;-CONVERT Macro Definition-;

```

convert      macro          string,value
        extrn   integer     ;;it is in our support library
        lxi     h,string
        call    integer
        shld    value
        endm

```

```

;-----
;-----

```

We now have everything that we need to look after our screen I/O, and it can now be reused in every program we write. It looks like a high-level language that we have written but, in fact, the code is much more compact than any language and, anyway, what is wrong with having one's own language? Naturally, we need not go the whole hog in using macros, but just use them occasionally. We are really only attempting to illustrate the use of macros and relocatable code.

Our 'printf' statement is like a normal string-output routine except that we can insert control characters or control strings, and we can insert the values of strings, character variables or string variables without having to think too hard about it.

```

printf      'The value of the answer is %d',ANS
           would print out as:-

```

The value of the answer is 254

if the variable ANS happened to contain 254. If we wanted it in hex, then it would be:-

```

printf      'The hex value of the answer is %x',ANS
           would print out as:-

```

The value of the answer is FE

We can insert strings, too.

```

printf      'Dear %s, what is %d plus %d?\n',NAME,FIRST,SECOND
           might print out as:-

```

Dear Lionel, what is 4 plus 3?

In fact, the following are recognized after a '%' sign.

d-----A signed decimal integer.  
u-----An unsigned decimal integer.  
x-----A Hexadecimal integer. (with leading zeros)  
c-----A character variable.  
s-----A string.

At present, it recognizes the following control characters:-  
\n-----Carriage-return/Line-feed sequence  
\t-----A tab  
\b-----Backspace  
\r-----Carriage-return  
\f-----Form Feed  
\\-----Backslash

Of course, in programs that use the screen, it would be useful to have a code that places the cursor at particular coordinates. In this way, the actual means of placing the cursor can be changed to accomodate other terminals merely by changing the macro or support routine.

If our program file is called TOHEX.ASM and our support library is called IO.ASM then our own little example is assembled by typing:-

**A-RMAC TOHEX! RMAC IO! LINK TOHEX,IO**

We have reached the stage of describing more formally the assembler directives of RMAC that exist over and above those of MAC.

ASEG	Use the absolute location counter.(set by an ORG statement)
CSEG	Use the code location counter. (the default condition)
DSEG	Use the data location counter. (location determined by the linker)
COMMON	Use the common location counter. (location determined by the linker)
PUBLIC	Declare the symbol as public so it can be referenced in another module
EXTRN	Declare that the symbol is defined in another module.
NAME	Name of the current module. (used for libraries).

We have not used ASEG or COMMON in our examples, and they are not really much needed. Otherwise they are best explained by there use in the above examples.

Digital Research distribute a great number of examples of Macro files, and the Z80.LIB file that allows RMAC to understand Z80 opcodes is on the CP/M+ disc. The others can be obtained from the CP/M User Group (UK). The CP/M User Group Library contains a large number of macro libraries constructed over the years by contributors, and these are also worth getting in order to learn about their use. I like to copy good macros and borrow good subroutines for later use. It is worth being methodical and documenting what you have carefully. It all saves work later on. Digital Research do a very good manual for MAC which explains in some detail about macros. It is worth getting hold of.

Very soon there comes a time when learning a new language or assembler when one must try it out, firstly by gingerly altering existing code and then getting ones head down and writing new code. If you are not familiar with RMAC, then perhaps the time is now.

## Appendix A – CP/M assemblers

### A.1 CP/M Assemblers

It is rather facile to consider assemblers without seeing them as part of a development package. After all, the simplest programs cannot be assembled and run without a linker or a loader, at least. Sooner or later you will have a program bug that requires a debugger. The simplest development system available under CP/M is the one supplied with CP/M 1.4 and 2.2 itself. The development package consists of :-

- ASM.COM - the assembler
- DDT.COM - the debugger
- LOAD.COM - the loader

This package originated within The Naval Postgraduate School, Monterey, California. Dr Kildall was responsible for at least a part of the original versions of the package. The assembler, loader and a proportion of the Debugger were written in PL/M. They have undergone a certain amount of modification over the years, and the assembler itself has been developed into Digital's RMAC, the relating macro assembler. The development system is indeed basic fare and was never intended for very complicated work. Programs have to be assembled to run at absolute locations and it is therefore impossible to develop large programs in a modular fashion. Large programs are tedious to develop as they have to be completely reassembled after every alteration. As there is no way of knowing where all the subroutines are, except by inspection of a listing '.PRN' file. Originally, the ASM.COM file was shipped with CP/M to facilitate its installation on a particular micro, rather than for use as a program development tool per se. It is surprising, therefore, just how much was developed using this tool. It has altered little over the years, but is reliable and virtually bug-free. It was intended that anyone who needed to do intensive assembler work should use

- MAC.COM - the macro assembler
- SID.COM - the Symbolic Instruction Debugger
- LOAD.COM - the same old loader

MAC is a fine tool for those who do not require relocatable object files. It comes with an excellent range of macro files to help with sequential I/O, provide instructions, provide structured assembly constructs, etc. Its strength is its macro facilities, and the documentation emphasises this to a great degree. There is every encouragement to use them. The assembler comes with macro files to allow the assembler to assemble Z80 opcodes; not using the Zilog mnemonics, but a version of the extended intel mnemonics developed by Neil Colvin at TDL.

SID was an advance on DDT. It could take a symbol table, as outputted by MAC or LINK, and one could then refer to those symbols instead of values or machine addresses during a debugging session. This means that, when debugging a subroutine, one could refer to it by name. It is much easier to understand code that made calls to other subroutines if displayed by SID using symbols instead of addresses (eg CALL .OUTPUT rather than CALL 3417). SID had other features, such as passpoints, that gave it a supremacy over every other debugger produced for the 8080/Z80 chips.

MAC is basically an extended version of ASM that was able to support Macros. Its Macro facilities are good, but not quite as comprehensive as others, such as M80 or PASM. Its great limitation was its inability to produce



'REL' files. This meant that large programs were tedious to develop. More recently, with the coming of PL1-80, MAC was updated to produce relocatable output.

```

RMAC.COM      - the relocatable macro assembler
LINK.COM      - the linking loader
SID.COM       - the same Symbolic debugger
(or ZSID.COM) -
LIB.COM       - the library utility
XREF.COM      - the cross-referencing utility.
```

RMAC is still descended from ASM. It is very similar to MAC but produces relocatable files. With CP/M plus, the installation of a CP/M system required the production of 'REL' files to link in with the operating system itself. If an assembler produces relocatable files, these files can be linked together to produce the final executable memory image, located at 0100H onwards. This gives immense advantages. Only one module needs to be developed at any time and so the time spent in assembly is vastly reduced. If one is editing a short module rather than an entire program, the amount of disc-work is less. All the commonly-used modules, such as data formatting, file i/o, screen-handling etc can be placed in a library file, which is searched by the linker at linkage time to resolve all external references.

SID is of limited use to debug relocatable files themselves, as it is unable to load them as object code and is unable to cope with addresses that are relative to a relocation base. It can, of course, use the symbol table that is output from LINK to debug the resulting '.COM' file.

The RMAC, ZSID, LINK, LIB and XREF combination is hard to beat. It is compatible with the Microsoft assembler and uses the Microsoft 'REL' format, and provides what is probably the best assembly development package on 8-bit micros under CP/M.

A separate line to Gary Kildall's development system was the TDL package. Neil Colvin was responsible for one of the first Z80 assemblers, and it certainly a product that grew into maturity. For some time it was tape-rather than disk-based, and it was a surprisingly long time before it was available under CP/M on disk. It appeared before MAC and was one of the first CP/M macro assemblers. It came with a linker and a debugger. This assembler positively bristled with features and is still a preferred tool with many Z80 assembly programmers. It eschewed the Zilog mnemonics for its own brand of Intel mnemonics. The pseudo-ops were quite different, and completely idiosyncratic. TDL's extended Intel mnemonics subsequently became more commonly used than Zilog's, and were easier to implement in assemblers and disassemblers. The manual for the assembler was thorough and complete, but required a familiarity with computer science. It was not easy. TDL disappeared suddenly and the software was then sold as

```

MACROII.COM
DEBUG.COM
LINK.COM
```

Neil Colvin moved to P.S.A. (Phoenix) and the Assembler reappeared with two new and powerful bedfellows that breathed new life into the product.

```

PASM.COM
BUG.COM
PLINK.COM
```

or (PLINK II.COM)

Undoubtedly, the Phoenix package is the most expensive, but most versatile and powerful development package available with CP/M, and will even support overlays. BUG and PLINK are remarkable products. PLINK has the most sophisticated methods of handling overlays of all the linkers and manages to link together modules developed using different '.REL' formats. It was also the first linker on CP/M that linked to disc, allowing the linkage of large '.COM' files. BUG is a maddening product, in that it has a range of useful advanced features, yet has a poor user-interface, requiring multiple keystrokes, where SID needs but one. It lacks passpoints, one of SID's best features, and cannot handle symbols. It gives the impression of being stopped in mid development, and now that the world has moved to 16-bit software, I suppose that BUGII will not now see the light of day. Phoenix now produce the best 16-bit development system, but have abandoned CP/M-86 for MSDOS.

The third great assembler came from Microsoft. This assembler is a good product that is useful for those who prefer the Zilog mnemonics. Its macro facilities are powerful, but it does not come with the macro libraries that are so useful with RMAC. The Linker is comparatively primitive when compared with PLINK.COM as it does not link disk to disk. Overlays are difficult to produce. M80 also suffers from the lack of a good Debugging tool. Nevertheless, M80 is probably the best known and most used relocating Macro assembler running under CP/M.

M80.COM	- The Macro Assembler
L80.COM	- The linker
LIB.COM	- The library manager
CREF.COM	- The cross-referencer

One should not neglect the final development package, the SD Systems package. Unlike the others mentioned, the SD Package sticks religiously to Z80 ops, pseudo ops and Macros. It is robust and useful, but does not produce relocatable code and lacks a debugger.

One of the most interesting and unusual development systems is the ML-80 package. This is a symbolic relocating macroassembler and linker. This is, fortunately, public domain software and uses completely different ops and pseudo-ops. It uses a syntax rather reminiscent of C, and can assemble statements that use such constructs as IF..GOTO and REPEAT...UNTIL. Source lines read more like a high-level language. lines such as:-

**A=M(7)+(B-8),-2;**

will compile into code identical with the more usual:-

```
LDA 0007
MWI B,8
ADD B
SUI 2
```

Its use has been surprisingly limited in spite of it being the only relocating macro assembler that is public property. The reason for its obscurity is that a good manual has never done the rounds and, if one appears, then we may see the package more widely used.

ML80	The general macro processor
L81	The structured assembly language parser
L82	code generator
L83	Linker

There are several other assemblers around. They mostly are available in the CP/M User library. The more important ones are as follows :-

**ASMX** -This assembler recognises extended intel Z80 mnemonics that are similar, but not the same as the TDL mnemonics. It seems to work quite well after one or two irritating little bugs have been cured. It is in US Vol 16

**MACASM** -For some reason, I have never got round to trying this assembler and I don't know anyone who has. It processes macros and recognises only 8080 mnemonics. It is in US Vol 16

**Z80ASM** -This appears in source form as well as in a '.COM' file. It works quite well but there are a few bugs. There is a much improved form which is in the CP/M user group (UK) library. The assembler recognises the Zilog mnemonics. It is found in US Vol 16 and an improved version is in UK Vol 20 as ZSM. This latter, by Neil Harrison, is thoroughly recommended.

**MILMON80** -This used to be given away free with Processor Technology Kit. It doesn't work very well at all so is only of historical value. It is one of those old Monitor-Editor-Assembler packages we used to like before we had floppy disks. It is in the US Vol 17

**RTMASM** -This assembler will assemble a series of assembly source files into one '.COM' file. It is 8080 only and seems to work well. It is in US Vol 32.

**LINKASM** - US Vol 36

## A.2 Intel Assembler Pseudoops

Generally, 8080 assemblers follow the conventions laid down by Intel in their ISIS development system.

<b>DB</b>	<b>exp(s) or string(s)</b>	Define 8-bit data byte(s). Expressions must evaluate to one byte.
<b>oplab:</b>	<b>DS      expression</b>	Reserve data storage area of specified length.
<b>oplab:</b>	<b>DW      exp(s) or string(s)</b>	Define 16-bit data word(s) Strings limited to 1 - 2 characters.
<b>oplab:</b>	<b>ELSE    null</b>	Conditional assembly. Code between else endif is assembled if expression in IF clause if false.
<b>oplab:</b>	<b>END      expression</b>	Terminate assembler pass. Prog execution starts at expression. If null then 0.
<b>oplab:</b>	<b>ENDIF    null</b>	Terminate conditional assembly block.

## Appendix A - CP/M Assemblers

name	EQU	expression	Define symbol 'name' with value 'expression'. Symbol is not redefinable.
oplab:	IF	expression	Assemble code between IF and following ELSE or ENDF directive if 'exp' is true.
oplab:	ORG	expression	Set location counter to 'expression'.
name	SET	expression	Define symbol 'name' with value 'expression' symbol can be redefined.

### A.3 Macro Directives

null	ENDM	null	Terminate macro definition
oplab:	EXITM	null	Alternate terminator of macro definition.
oplab:	IRP	dummy param <list>	Repeat instruction sequence substituting one character from list for dummy param in each iteration.
oplab:	IRPC	dummy param, text	Repeat instruction sequence substituting one character from text for dummy param in each iteration.
null	LOCAL	label name(s)	Specify label(s) in macro definition to have local scope.
name	MACRO	dummy param(s)	Define macro 'name' and dummy parameters to be used in macro definition.
oplab:	REPT	expression	Repeat rept block 'expression' times.

### A.4 Relocation Directives

## Appendix A - CP/M Assemblers

oplab:	ASEG	null	Assemble subsequent instructions and data in the absolute mode.
oplab:	CSEG	boundary specification	Assemble subsequent instructions and data in relocatable mode using data location intr.
oplab:	DSEG	boundary specification	Assemble subsequent instructions and data in relocatable mode using data location intr.
oplab:	EXTERN	name(s)	Identify symbols used in this programme module, but defined in a different module.
oplab:	NAME	module name	Assigns a name to a program module.
oplab:	PUBLIC	name(s)	Identify symbols defined in this module that are to be available to other modules.
oplab:	STKLN	expression	Specify the number of bytes to be reserved for the stack for this module.

## A.5 Macro-80

M80, the Microsoft assembler, is the most complete assembler for CP/M. It is expensive and, now that RMAC is issued with CP/M+, not worth the extra expense. For the professional programmer it is, however, essential; it is amusing to see that one or two assembler listings to have emanated from Digital Research have been written for M80 and not RMAC. We have mentioned MAC and RMAC in the chapter on the CCP. M80, being a foreign product, we will describe here

M80 (MACRO-80) is invoked by the following command --

M80 Relfile,prnfile=source

where

relfile is the device/filename for the object program  
prnfile is the device/filename for the listing  
source is the device/filename for the source

The following switches may be specified in the command line --

O Print all listing addresses in octal  
H Print all listing addresses in hexadecimal  
  
R Force generation of an object file  
L Force generation of a listing file  
C Force generation of a cross reference file  
  
Z Assemble Zilog (Z80) mnemonics  
I Assemble Intel (8080) mnemonics

P Each /P allocates an extra 256 bytes of stack space for use during assembly. Use /P if stack overflow errors occur during assembly.

:MACRO-80 Pseudo-Ops

The following are the pseudo-ops recognized by MACRO-80 --

ASEG	COMMON	CSEG	DB	DC
DS	DSEG	DW	END	ENTRY
PUBLIC	EQU	EXT	EXTRN	NAME
ORG	PAGE	SET	SUBTTL	TITLE
.COMMENT	.PRINTX	.RADIX	.REQUEST	.Z80
.8080	IF	IFT	IFE	IFF
IF1	IF2	IFDEF	IFNDEF	IFB
IFNB	ENDIF	.LIST	.XLIST	.CREF
.XCREF	REPT	ENDM	MACRO	IRP
IRPC	EXITM	LOCAL	COND	ENDC
*EJECT	DEFB	DEFS	DEFW	DEFM
DEFL	GLOBAL	EXTERNAL	INCLUDE	MACLIB
ELSE	.LALL	.SALL	.XALL	

We cannot describe the meanings of all these pseudo-ops here; it is best to use the manual.

The following are the assembler error codes. Mercifully, these are shown on the screen so there is not need to thumb through interminable '.PRN' file listings to find errors.

A	Argument Error	O	Bad opcode or objectionable syntax
C	Conditional nesting err	P	Phase error
D	Double Defined Symbol	Q	Questionable
E	External error	R	Relocation
M	Multiply Defined Symbol	U	Undefined symbol
N	Number error	V	Value error

You are also likely to see some of the following messages in the course of program development!

No end statement encountered in input file  
-- no END statement

Unterminated conditional  
-- at least one conditional is unterminated

Unterminated REPT/IRP/IRPC/MACRO  
-- at least one block is unterminated

[xx] [No] Fatal error(s) [,xx warnings]  
-- the number of fatal errors and warnings

## A.6 Link-80

L80 (LINK-80), the Microsoft linker, is not so good as the Digital Research linker. Part of the reason for the great popularity for the Phoenix assembler PLINK was that it would substitute for the rather mediocre LINK-80. The latter does its linkage in memory, so there are considerable limitations on the size of program that it can link. It is, however, reasonably fast.

Each command to LINK-80 consists of a number of filenames and switches separated by commas --  
relfile1/switch,relfile2/switch,relfile3/switch....

Where relfilex is a CP/M valid filename (see the chapter on the CCP)

If the input device for a file is omitted, the default is the currently logged disk. If the extension of a file is omitted, the default is '.REL'. After each line is typed, LINK-80 will load or search the specified files, and, when finished, it will list all symbols that remain undefined followed by an asterisk. LINK-80 is invoked by the program name 'L80'.

LINK-80 can be used to generate a '.COM' file of a FORTRAN-80, or BASCOM program. This can be done by typing --

**A>L80 program/E**

LINK-80 will respond with a string of the form --  
[aaaa bbbb nn]

The user may then create the '.COM' file by typing --  
SAVE nn program.COM

The following are the switches for LINK-80. These switches are preceeded by a slash (/).

Switch	Function
--------	----------

```

-----
/R      Reset -- put loader back in initial state

/E or /E:Name
        Exit LINK-80 and return to CP/M. Search the system library for
any undefined references. /E:Name uses Name for the start address of the
program.

/G or /G:Name
        Start execution of the program. Again, if /G:Name is specified,
Name defines the start address of execution.

<filename>/N
        Save the binary on disk under the name 'filename.COM'.

/P:adr and /D:adr
        Set the Program and Data area origins for the next program to be
loaded.

/U
        List the origin and end of the program and data area as well as
all undefined globals.

/M
        List the origin and end of the program and data area, all
defined globals and their values, and all undefined globals followed by an
asterisk.

<filename>/S
        Search 'filename.REL' to satisfy references.

/X
        If a filename/N was specified, /X will cause the file to be
saved in Intel HEX format with a extension of '.HEX'.

/Y
        If a filename/N was specified, /Y will create a filename.SYM
file when /E is entered. This file contains the names and addresses of all
globals for use with SID or ZSID.

        Three numbers are specified after the /E and /G switches are executed.
They are given in the form --
        [aaaa bbbb nn]
aaaa - start address of program
bbbb - address of next available byte
nn   - number of 256-byte pages used

```



## A.7 Lib-80

LIB-80 is the Microsoft '.REL' file library maintenance utility. It is not so well known as the digital Research utility but appears to work properly. For the assembly-language programmer, a library manager is essential, as program development will involve building up a library of modules. Linking with a library file manager is much more convenient as giving the linker a long command line, and only those modules that are actually used are accessed.

To invoke LIB-80, type --  
A>LIB

Commands to LIB-80 consist of an optional destination filename which sets the name of the library being created, followed by an equal sign, followed by module names separated by commas or filenames with module names enclosed in angle brackets separated by commas.

To select a given module from a file, use the name of the file followed by the module(s) specified enclosed in angle brackets and separated by commas. If no modules are selected from a file, then all modules are selected.

```
/O Set listing radix to Octal
/H Set listing radix to Hexadecimal
/U List undefineds
/L List cross reference
/C Create -- start LIB over
/E Exit -- rename '.LIB' to '.REL' and exit
/R Rename -- rename '.LIB' to '.REL'
```

## A.8 CREF-80

Microsoft's CREF-80 is a '.MAC' file cross-referencer. It requires a '.PRN' file as produced by M80 and will provide a cross-referenced listing that gives the line reference for the place where a symbol (variable, location etc) was defined and where it was used. It is very useful for pruning and tidying source files and for clearing out subroutines that never get called. It is not essential, merely useful.

CREF-80 is invoked by typing --

A>CREF80 listfile=source

where 'listfile' is the name of the file 'listing.LST' generated and 'source' is the name of the file 'source.CRF' generated by MACRO-80. 'listing' is optional, and, if omitted, the listing file generated is named 'source.LST'.

Whereas RMAC and MACRO-80 are fairly straightforward implementations of the intel assembler, The Phoenix assembler (PASM or MACRO-II) is maverick in several ways. For a start, the pseudo-ops are quite unlike the Intel standard. Here are the main Pseudo-ops with their M80 equivalent and the Zilog version where they occur. Note that there are many more PASM codes not listed which give a great deal of control over the assembly. Note also that the PASM .IFx series have a different syntax to the M80 version so we have put them in delimiters.

## A.9 Assembler Pseudo-Ops

M80	Zilog	CDL/PASM
ASEG		.PABS
COMMON		.LOC
CSEG		.PREL
DB	DEFB DEFM	.ASCII or .BYTE
DC		.ASCIS
DS	DEFS	.BLKB
DSEG		.LOC .DATA.
DW	DEFW	.WORD
END		.END
ENTRY/PUBLIC	GLOBAL	.ENTRY
EQU		=
EXT EXTRN	EXTERNAL	.EXTERN
INCLUDE \$INCLUDE MACLIB		.INSERT
NAME		.IDENT
ORG		.LOC
PAGE	*EJECT	.PAGE
SET	DEFL	=
SUBTTL		.SBTTL
TITLE		.TITLE
.COMMENT		REMARK
.PRINTX		.PRINTX
.RADIX		.RADIX
.Z80		.Z80
.8080		.I8080
.REQUEST		
IF/IFT	COND	<.IF>
IF[- IFF		<.IFL>
IF1		<.IF1>
IF2		<.IF2>
IFDEF		<.IDDEF>
IFNDEF		<.IFNDEF>
IFB		<.IFB>
IFNB		<.IFNB>
IFIDN		<.IFIDN>
IFDIF		<.IFDIF>
ELSE		< >
ENDIF	ENDC	]
.LIST		
.XLIST		
.SFCOND		
.TFCOND		
.PHASE		
.DEPHASE		
REPT - ENDM		
IRP - ENDM		
IRPC - ENDM		
MACRO		<.DEFINE>
EXITM		.EXIT

LOCAL

#### A.10 PASM

PASM is invoked by the command:-

**A>PASM filename.typ d: xx**

where filename.typ is a valid CP/M filename with an optional filetype (defaults to '.ASM'). The object files can be directed to a drive other than the one specified by the source filename by d:. One can follow all this by conditional switches as follows:-

- A List every text character processed
- B Send the '.PRN' listing file to both disc and printer
- C List all listing control pseudo-ops.
- D Send the listing to disc only.
- H Produce a '.HEX' file output.
- I Send every byte assembled to the '.PRN' file rather than a max of 6. per line
- L Suppress the output of linkage information.
- O Produce object only, no listing
- P Send symbol table to the end of the object file.
- S Suppress macro expansion.
- X Suppress all listing information (until an error occurs)
- Y Suppress output of the symbol table.

If an error occurs, a character describing the error is sent to the listing or '.PRN' file and a '?' is placed to the right of where the error occurred. If the 'X' conditional switch is used and the listing diverted to the console using DEVICE, then only the lines in error are printed to the console. Error codes are:-

- A Argument error (uncategorizable error).
- B Bad macro definition or macro call.
- D Duplicate symbol.
- E External symbol error.
- I Internal symbol error.
- L Label error.
- M Multiple definition of symbol.
- O Operation error.
- P Phase error.
- Q Questionable error. (ambiguity in source file).
- R Relocation error.
- S Subscript error. (it has exceeded the array space available)
- T Table overflow.
- U Undefined label/symbol error.
- X Index error (IX or IY)
- Z Z80 warning. whilst the .8080 op is in force, a Z80 opcode has been found.
- \* A user-defined macro error.

The PLINK and PLINKII linker have too many options to allow description in this chapter. The reader is referred to the PLINK manual. Complex linkages, particularly if involving overlays, are best done by providing '.LNK' files or running '.SUB' files. A simple '.REL' file as produced by PASM, or any assembler producing Microsoft '.REL' files, is linked simply by typing:-

PLINK relname

Which will produce a '.COM' file of the same name.

As the XITAN/TDL/PASM/CDL assemblers use non-standard assembler opcodes for the Z80 opcodes it is worth listing them here.

### A.11 TDL-Xitan Ops

#### A.11.1 8-bit Load Group

```
MOV    r,r'      r <- r'
        Move to register from alternate register
MOV    r,m        r <- (HL)
        Move the contents of location addressed by HL into register (ABCDEH or L)
MOV    r,d(ii)    r <- (ii+d)
        Move data to register from indexed memory location (ii + d)
MOV    m,r        (HL) <- r
        Move data to location addressed by HL from the register r (ABCDEH or L)
MOV    d(ii), r   (ii + d) <- r
        Load indexed addressed memory (ii + d) from register r (ABCDEH or L)
MVI    r, n       r <- n
        Move immediate data into register (ABCDEHL)
MVI    m,n        (HL) <- n
        Move immediate data into memory location addressed by HL
MVI    d(ii), n   (ii + d) <- n
        Move immediate data into location addressed by (ii + d)
LDA    nn         A <- (nn)
        Load accumulator from the memory location addressed by (nn)
STA    nn         (nn) <- A
        Store accumulator in the memory location addressed by (nn)
LDAX   zz         A <- (zz)
        Load the accumulator from memory location addressed by BC or DE (B,D)
STAX   zz         (zz) <- A
        Store the accumulator in memory location addressed by BC or DE (B,D)
LDAI   A <- I
        Load accumulator from interrupt vector register I
LDAR   A <- R
        Load accumulator from Memory Refresh register R
STAI   I <- A
        Store accumulator in interrupt vector register I
STAR   R <- A
        Store accumulator in Memory Refresh register R
```

#### A.11.2 16-bit Load Group

```
LXI    rr, nn     rr <- nn
        Load extended immediate into register pair rr (B D H SP)
LXI    ii, nn     ii <- nn
        Load extended immediate nn to index register (3Y, X)
LBCD   nn         B <- (nn+1), C <- (nn)
        Load BC direct from memory locations addressed by nn
LDED   nn         D <- (nn + 1) E <- (nn)
        Load DC direct from memory locations addressed by nn
LHLD   nn         H <- (nn + 1) L <- (nn)
        Load HL direct from memory locations addressed by nn
LIXD   nn         IX/H <- (nn + 1) IX/L <- (nn)
        Load IX direct from memory locations addressed by nn
LIYD   nn         IY/H <- (nn + 1) IY/L <- (nn)
```

# Appendix A - CP/M Assemblers

```

Load IY direct from memory locations addressed by nn
LSPD  nn      SP/H <- (nn + 1) SP/L <- (nn)
Load SP direct from memory locations addressed by nn
SBCD  nn      (nn + 1) <- B (nn) <- C
Store contents of BC direct into memory locations addressed by nn
SDED  nn      (nn + 1) <- D (nn) <- E
Store contents of DE direct into memory locations addressed by nn
SHLD  nn      (nn + 1) <- H (nn) <- L
Store contents of HL direct into memory locations addressed by nn
SIXD  nn      (nn + 1) <- IX/H (nn) <- IX/L
Store contents of IX direct into memory locations addressed by nn
SIYD  nn      (nn + 1) <- IY/H (nn) <- IY/L
Store contents of IY direct into memory locations addressed by nn
SSPD  nn      (nn + 1) <- SP/H (nn) <- SP/L
Store contents of SP direct into memory locations addressed by nn
SPHL  SP <- HL
Fill the stock pointer with the contents of HL
SPIX  SP <- IX
Fill the stock pointer with the contents of IX
SPIY  SP <- IY
Fill the stock pointer with the contents of IY
PUSH  qq      (SP - 1) <- qq/H (SP - 2) <- qq/L, SP <- SP - 2
Push the register pair (B C H PSW) on to the stack
PUSH  ii      (SP - 1) <- ii/H (SP - 2) <- ii/L, SP <- SP - 2
Push the index register on to the stack (X, Y)
POP   qq      qq/H <- (SP + 1) qq/L <- (SP) SP <- SP + 2
Pop into the register pair (B D H PSW) from the stock
POP   ii      (ii/H <- (SP + 1), ii/L <- (SP) SP <- SP + 2
Pop into the index register pair (X, Y) from the stock

```

## A.11.3 Exchange, block transfer and search group

```

XCHG          HL <-> DE
              Exchange the HL and DE registers
EXAF          PSW <-> PSW'
              Exchange accumulator and flags with alternate registers
EXX           BCDEHL <-> B'C'D'E'H'L'
              Exchange general purpose registers with alternate set

XTHL          H <-> (SP + 1) L <-> (SP)
              Exchange HL with the top of the stack
XTIX          IX/H <-> (SP + 1) IX/L (SP)
              Exchange IX with the top of the stack
XTIY          IY/H <-> (SP + 1) IY/L (SP)
              Exchange IY with the top of the stack

LDI           (DC) <- (HL) DE <- DE + 1, HL <- HL + 1 BC = BC - 1
              Block load with increment (transfer)
LDIR          LDI until BC = 0
              Repeating block load with increment (transfer)
LDD           (DC) <- (HL) DE <- DE - 1, HL <- HL - 1 BC = BC - 1
              Block load with decrement (transfer)
LDDR          repeat LDD until BC = 0
              Repeating block load with decrement (transfer)

CCI           A - (HL) HL <- HL + 1, BC <- BC - 1
              Compare with increment
CCIR          repeat CCI until A = (HL) or BC = 0
              Block compare with increment until a match or BC = 0
CCD           A - (HL) HL <- HL - 1, BC <- BC - 1
              Compare with decrement
CCDR          repeat CCD until A = (HL) or BC = 0
              Block compare with decrement until a match or BC = 0

```

## A.11.4 8-bit arithmetic and logical group

```

ADD    r      A <- A + r
              Add accumulator with register r
ADD    m      A <- A + (HL)
              Add accumulator with indirectly addressed memory location (HL)
ADD    d(ii)  A <- A + (ii + d)
              Add accumulator with indexed addressed memory location (ix + d)
ADI    r      A <- A + n
              Add accumulator with immediate data n
ADC    s      A <- A + S + CY
              Add accumulator and specified operand with carry (r, Mer(ii))
ACI    r      A <- A + n + CY
              Add accumulator with immediate n
SUB    s      A <- A - S
              Subtract operand S(ABCDEHL, d(ii) or n) from accumulator
SUI    n      A <- A - n
              Subtract immediate n from accumulator
SBB    s      A <- A - S - CY

```

## Appendix A - CP/M Assemblers

```

    Subtract with borrow acc and specified operand (ABCDEHLM d(ii) or n)
SBI    n            A <- A -n - CY
    Subtract with borrow accumulator and immediate data n
ANA    s            A <- A & S
    Logical "and" accumulator with operand (ABCDEHL,M or d (ii) ))
ANI    n            A <- A & n
    Logical "and" accumulator with immediate n
ORA    s            A <- A ! s
    Logical "or" accumulator with operand (ABCDEHLM or d (ii))
ORI    n            A <- A ! n
    Logical "or" accumulator with immediate data n
XRA    s            A <- A ^ S
    Logical "exclusive or" accumulator and operand (ABCDEHLM or d(ii))
XRI    n            A <- A ^ n
    Logical "exclusive or" accumulator and immediate n
CMP    s            A - S
    Compare operand s(ABCDEHLM or d(ii)) to accumulator
CPI    n            A - n
    Compare immediate data n with accumulator
INR    s            S <- S + 1
    Increment register (ABCDEHLM or d(ii))
DCR    s            S <- S - 1
    Decrement register (ABCDEHLM or d(ii))

```

## A.11.5 General-purpose Arithmetic and control group

DAA                   convert A to packed BCD after add or subtract of  
                           packed BCD  
 CMA                   A <- ~A  
           Complement the bits in accumulate  
 NEG                   A <- -A  
           Negative accumulator  
 CMC                   CY <- ~CY  
           Complement the carry flag  
 STC                   CY <- 1  
           Set the carry flag to 1  
 NOP                   no operation  
           No operation  
 HLT                   halt  
           Halt the processor  
 DI                    IFF <- 0  
           Disable interrupts  
 EI                    IFF <- 1  
           Enable interrupts  
 IM0                   interrupt mode 0  
           Interrupting device must insert 1 instruction on to data bus for  
           execution  
 IM1                   interrupt mode 1  
           A reset 0038H instruction will be executed on interrupt  
 IM2                   interrupt mode 2  
           Interrupting device must insert 1 byte on to data bus which will be  
           combined with the interrupt vector to create a call instruction for  
           execution.

## A.11.6 16-Bit Arithmetic Group

DAD   rr            HL <- HL + rr  
           Double precision add register pair to HL  
 DADC   rr           HL <- HL + rr + CY  
           Double precision add with carry register pair to HL  
 DSBC   rr           HL <- HL - rr - CY  
           Double precision subtract with carry register pair from HL  
 DADX   tt           IX <- IX + tt  
           Double precision add register pair to IX  
 DADY   uu           IY <- IY + uu  
           Double precision add register pair to IY  
 INX    rr           rr <- rr + 1  
           Increment register pair (double precision)  
 INX    ii           ii <- ii + 1  
           Increment index register (double precision)  
 DCX    rr           rr <- rr - 1  
           Decrement register pair (double precision)  
 DCX    ii           ii <- ii - 1  
           Decrement index register (double precision)

## A.11.7 Skew (rotate and shift) group.



```

RLC
    Rotate left circular accumulator
RAL
    Rotate accumulator left (through carry)
RRD
    Rotate right circular accumulator (with branch carry)
RAR
    Rotate accumulator right
RLCR s
    Rotate left circular register (with branch carry)
RALR s
    Rotate left (through carry) register
RRCR s
    Rotate right circular register (with branch carry)
RARR s
    Rotate right (through carry) register
SLAR s
    Shift left arithmetic register
SRAR s
    Shift right arithmetic register
SRLR s
    Shift right logical register (halve it)
RLD
    Rotate left decimal
RRD
    Rotate right decimal

```

#### A.11.8 Bit, set, reset, and test group

```

BIT    b, r      ZF <- £r [b]
        Test bit b of register r
BIT    b, m      ZF <- £(HL)[b]
        Test bit b of indirectly addressed memory location (HL)
BIT    b, d (ii) ZF <- £ (ii + d) [b]
        Test bit b of indexed addressed memory location
SET    b, s      s [b] <- 1
        Set bit b of operand (ABCDEHLM, d (ii))
RES    b,s      s[b] <- 0
        Reset bit b of operand (ABCDEHLM, d(ii))

```

## A.11.9 Jump Group

```

JMP  nn      PC <- nn
      Jump to address nn
JZ   nn      if zero then jump else continue
      Jump to address nn if zero
JNZ  nn      if not zero
      Jump to address nn if not zero
JC   nn      if carry
      Jump to address nn if not carry
JNC  nn      if not carry
      Jump to address nn if not carry
JPO  nn      if parity odd
      Jump to address nn if parity odd
JPE  nn      if parity even
      Jump to address nn if parity even
JP   nn      if sign positive
      Jump to address nn if sign positive
JM   nn      if sign negative
      Jump to address nn if sign negative
JO   nn      if overflow
      Jump to address nn if overflow
JNO  nn      if not overflow
      Jump to address nn if not overflow
JMPR nn      PC <- nn where -128 < nn - PC < 129
      Jump n relative to address
JRZ  nn      if zero, then JMPR else continue
      Jump n relative if zero
JRNZ nn      if not zero
      Jump n relative if not zero
JRC  nn      if carry
      Jump n relative if carry
JRNC nn      if no carry
      Jump n relative if no carry
DJNZ nn      B <- B - 1 if B = 0 then continue else JMPR
      Decrement B and jump e relative on no zero
PCHL nn      PC <- HL
      Jump to location addressed by HL
PCIX nn      PC <- IX
      Jump to location addressed by IX
PCIY nn      PC <- IY
      Jump to location addressed by IY

```

## A.11.10 Call Group

```

CALL  nn      (SP-1) <- PC/H, (SP-2) <- PC/L PC <- nn SP <- SP-2
      Call subroutine at address nn
CZ    nn      if zero, then call else continue
      Call subroutine if zero
CNZ   nn      if not zero, then call else continue
      Call subroutine if not zero
CC    nn      if carry, then call else continue
      Call subroutine if carry
CNC   nn      if not carry, then call else continue
      Call subroutine if not carry
CPO   nn      if parity odd, then call else continue
      Call subroutine if parity odd
CPE   nn      if parity even, then call else continue
      Call subroutine if parity even
CP    nn      if sign positive, then call else continue
      Call subroutine if sign positive
CM    nn      if sign negative, then call else continue
      Call subroutine if sign negative
CO    nn      if overflow, then call else continue
      Call subroutine if overflow
CNO   nn      if not overflow, then call else continue
      Call subroutine if not overflow

```

### A.11.11 Return Group

```

RET          PC/H <- (SP + 1), PC/L <- (SP) SP <- SP + 2
             Return from subroutine
RZ           if zero, then RET else continue
             Return from subroutine if zero
RNZ          if not zero, then RET else continue
             Return from subroutine if not zero
RC           if carry, then RET else continue
             Return from subroutine if carry
RNC          if no carry, then RET else continue
             Return from subroutine if no carry
RPO          if parity odd, then RET else continue
             Return from subroutine if parity odd
RPE          if parity even, then RET else continue
             Return from subroutine if parity even
RP           if plus, then RET else continue
             Return from subroutine if plus
RM           if minus, then RET else continue
             Return from subroutine if minus
RO           if overflow, then RET else continue
             Return from subroutine if overflow
RNO          if not overflow, then RET else continue
             Return from subroutine if not overflow
RETI         return from interrupt
             Return from interrupt
RETN         return from non-maskable interrupt
             Return from non-maskable interrupt
RST n        (SP-1) <- PC/H (SP-2) <- PC/L PC <- 8 xn
             where 0 <- n < 8

Restart at p
    
```

## A.11.12 Input and Output Group.

```

IN      n          A <- 1n
        Load accumulator from input port N
INP     r          r <- 1 (C)
        Load register r from port (C)
INI     (HL) <- 1(C) B = B - 1, HL <- HL + 1
        Input with increment
INIR    repeat INI until B = 0
        Block input with increment
IND     (HL) <- 1 (C) B <- B - 1, HL <- HL - 1
        Input with decrement
INDR    repeat IND until B = 0
        Block input with decrement

OUT     n          On <- A
        Output accumulator to peripheral port N
OUTP    r          O (C) <- r
        Output register r to port C
OUTI    O (C) <- (HL) B = B - 1, HL <- HL + 1
        Output with increment
OUTIR   repeat OUTI until B = 0
        Block output with increment
OUTD    O (C) <- (HL) B = B - 1 HL <- HL - 1
        Output with decrement
OUTDR   repeat OUTD until B = 0, Block output with decrement

```

## Appendix B – Introduction to BASIC-E

BASIC-E is really CP/M's own BASIC. It is still widely used, and there now exists upgraded versions in CBASIC (the pseudo-compiler) and CB-80 (the compiler). As it is in the public domain, and is not well documented, we will discuss it in some detail.

BASIC-E was developed in 1976 at the Microcomputer laboratory, Naval Postgraduate School, Monterey, California. The school was responsible for much of the CP/M software including parts of CP/M itself, TED (the friendly text editor), the remarkable ALGOL-M, and BASIC-E. The involvement of the school was not surprising in view of its geographical location near INTEL (who awarded generous bursaries to the school) and the presence of Gary (later Professor Gary) Kildall, who wrote CP/M whilst working for the school and for INTEL. Gordon Eubanks' supervisor for this project, which was part of his masters thesis, was Gary Kildall, who also converted it to the resident PL/M. BASIC-E was designed to be compatible, wherever possible, with the proposed ANSI standard and was converted to run under CP/M, which was being refined and developed at the same time at the school.

In order to allow BASIC-E to be used in the rather restricted memory size of micros at that time, it was designed in two parts, a table-driven parser which checks statements for correct syntax and generates code for a hypothetical stack machine, and a 'Run Time Monitor' (interpreter) which ran the intermediate code. It was not a true compiler in that it did not produce machine-readable code, yet it lacked the advantages of an 'interactive' interpreter as is most commonly found. However it ran happily in a 16k system and was free! Commercial software houses liked it because they were able to provide INT files rather than source programs. It was slow, but robust and accurate.

The BASIC-E language features include sequential and random diskette file capabilities, a wide range of predefined functions, user-defined functions, and strings up to 255 characters long. Arrays may have any number of subscripts and may be string or numeric. In a string array, each element can be a string of up to 255 characters. Numbers in BASIC-E are represented internally in binary floating point and approximately 7 significant decimal digits are carried. BASIC-E has no provision for formatted printing, and no provision for calling machine language subroutines.

One of the reasons for BASIC-E's surprisingly slow execution is its high numerical accuracy. It has a very good floating-point package which followed the proposed ANSI standard of maintaining seven digit integer precision. Floating point arithmetic numbers are represented in 32 bits with an 98 bit exponent, one bit sign and twenty-four bits of fraction. This provides slightly more than seven decimal digits of significance. (range 2.7E-39 to 3.6E38) To keep things simple, the choice of number type and accuracy is restricted but the processor is, therefore, kept busy with redundant number-crunching on floating point numbers that would have been better represented as integers.

Multidimensional Arrays are provided for numbers and strings (variable length strings and a dimensional arrays are both dynamically allocated). and there are logical operators. Several string handling routines are included. Both random and sequential file access is allowed.

Version 1 had several bugs which fouled up certain file operations, string

processing, and some of the predefined functions! The most intractable and harmful problem was a tendency for the seventh digit of precision to be erratic. Version 2 fixed most of the bugs and 'solved' the integer problem by printing only six of the seven digits maintained internally! Structured Systems Group finally tracked down the elusive integer bug and issued version K2.0. By this time BASIC-E had caught on and was being widely distributed. Various versions were in circulation, including the faulty version 1. There was a demand for a larger expanded version which could cope with commercial file handling applications and manage more sophisticated string handling operations. Gordon Eubanks, who was evidently now a naval officer on board a nuclear submarine, founded Compiler Systems to market the new updated BASIC-E as CBASIC (university developed software is public domain in the U.S.) CBASIC was much cleverer, with its ability to chain, interface with machine language subroutines etc, but retained the essential organization of BASIC-E. In spite of its ability to allow programs to specify integers, it tended to run even slower than BASIC-E. Despite its minor flaws, it quickly gained a reputation for its accuracy and good file-handling characteristics. Compiler Systems seems to have been little more than an office room with a telephone at this time. Users who were surprised by the length of time taken by the firm to answer technical queries were unaware that they had to wait until Gordon Eubanks' nuclear submarine surfaced before he could be contacted.

CBASIC grew in popularity and, in its present stability and maturity, remains CP/M's premier BASIC. Several microcomputer manufacturers issue it as standard with their Kit. It can even be bought to run on the TRS-80, and is now issued to run on the 8086 chip with CP/M-86. A true compiler, that compiles source code to Z-80 code, is to be launched at any moment. CBASIC is definitely evolved directly from BASIC-E and BASIC-E programs will run with CBASIC. Anyone who disassembles CBASIC will see whole sections from the old version and other parts which are completely unused in the modern revision. It is, however a trusty, debugged, and rugged piece of software that has stood the test of time. A great deal of software has been written for it, particularly in the commercial sphere, and there are direct program generators written for it (PEARL and T.L.O.) ; There even exists an S100 card that interprets the hypothetical stack INT file directly, thus speeding execution greatly.

The entire PL/M source of BASIC-E exists in the public domain. Be careful to get the completely debugged 7 digit integer version. I do not know of a friendly manual for BASIC-E but the original one, with all its formal language, is freely available. The CBASIC manual is very good, and worth buying if only to help you to understand BASIC-E. A good example of a BASIC-E program is OTHELLO.BAS (U.S.usergroup library Vol 5.14).

The source of BASIC-E is on U.S.U.G.lib vol 29-30

Copies of EBASIC.COM and ERUN.COM are to be found on U.S.U.G.lib vol 5

Sample games and programs, mostly rather silly but fun, are on U.S.U.G.lib vols 3,5,13,20, and 28.

Copies of the original manual, as well as a good version of EBASIC are obtainable from,

ComCen Microcomputers Ltd  
45-46 Wychtree Street,  
Morrison,  
SWANSEA SA6 8EX

Compiler Systems live at:-  
Compiler Systems INC,  
Post office box 145,  
Sierra Madre,  
California, 91024

They provide configurations of CBASIC for CP/M, MP/M, and CP/NET. They also are said to publish a software directory over 300 accounting and vertical market packages.



## B.1. BASIC-E commands

BASIC-E has the following commands and operators:--

```
ABS ASC ATN CHR$ CLOSE COS COSH DATA DEF DIM END FILE
FOR FRE FN GOSUB GOTO IF IFEND INP INPUT INT LEFT$ LEN
LET LOG MIDS ON NEXT OUT POS PRINT RANDOMISE READ REM
RESTORE RETURN RIGHT$ RND SGN SIN SINH STR$ SQR TAB
STEP STOP TAN THEN TO VAL
```

BASIC-E has the following expression elements:--

```
( ) ^ * / + - < > <= >= => =< <> LT LE GT GE EQ
NE NOT AND OR XOR
```

Commands and operators must be preceded and followed by either a special character or a space. Spaces may not be embedded within reserved words. Unless compiler toggle D is set, lower-case letters are converted to uppercase prior to checking to see if an <identifier> is a reserved word.

All BASIC-E statements are terminated by a carriage return or a semi-colon.

BASIC-E allows you to miss line numbers out if the line is unreferenced from elsewhere. They are ignored by the compiler except when they appear in a GOTO, GOSUB, or ON statement. In these cases, the line number must appear as the label of one and only one statement in the program. The line numbers can be in any order you like and can be real rather than integer. Line numbers may contain any number of digits but only the first 31 are considered significant by the compiler.

It does not allow symbols or expressions instead of line numbers, which is a shame (TBASIC does!) Variables and other identifiers can be long names such as something like THE.AMOUNT.THAT.YOU.OWE so programs are easy to understand. An identifier begins with an alphabetic character followed by any number of alphanumeric characters, or full-stops. Only the first 31 characters are considered unique. If the last character is a dollar sign the associated variable is of type string, otherwise it is of type floating point. All lowercase letters appearing in an identifier are converted to uppercase unless compiler toggle D is set to off. Other than these long identifiers, and the optional omission of line numbers, BASIC-E is fairly standard, and many BASIC programs from other interpreters will run without drastic alteration.

A variable in BASIC-E may either represent a floating point number or a string depending on the type of the identifier. Subscripted variables must appear in a DIM statement before being used as a variable.

A constant may be either a number or a string. All numeric constants are stored as floating point numbers. Strings may contain any ASCII character.

Numeric constants may be either a signed or unsigned integer, decimal number, or expressed in scientific notation. Numbers up to 31 characters in length are accepted but the floating point representation of the number maintains approximately seven significant digits (1 part in 16,000,000). The largest magnitude that can be represented is approximately 2.7 times ten to the minus 39th power.

eg:

10

-100.75639E-19

String constants may be up to 255 characters in length. Strings entered from the console, in a data statement, or read from a disk file may be either enclosed in quotation marks or delimited by a comma. Strings used as constants in the program must be enclosed in quotation marks.

eg:

"THIS IS THE ANSWER"

Expressions consist of algebra in combinations of variables, constants, and operators. The hierarchy of operators is:

- 1) ()
- 2) ^
- 3) \*, /
- 4) +, -, concat (+), unary +, unary -
- 5) relational ops <, <=, >, =, <> LT, LE, GT, GE, EQ, NE
- 6) NOT
- 7) AND
- 8) OR, XOR

Relational operators result in a 0 if false and -1 if true, NOT, and AND, and OR are performed on 32 bit two's complement binary representation of the integer portion of the variable. The result is then converted to a floating point number. String variables may be operated on by relational operators and concatenation only. Mixed string and numeric operations are not permitted.

EXAMPLES:

```
X + Y
AS + BS
(A <= B) OR (CS > DS) / (A - B AND D)
```

A program consists of one or more properly formed BASIC-E statements. An END statement, if present, terminates the program and additional statements are ignored. The entire ASCII character set is accepted.

Writing a program with BASIC-E consists of three steps. First the source program must be created on disk. Next the program is compiled by executing the BASIC-E compiler with the name of the source program provided as a parameter.

A>BASIC progname

Finally the intermediate (INT) file created by the compiler may be interpreted by executing the run-time monitor, again using the source program name as a parameter.

A>RUN progname

The source program must have a file type '.BAS'. The BASIC-E statements are free format, with the restriction that when a statement is not completed on a single line, a continuation character (\) must be that last character on the line. Spaces may precede statements and any number of spaces may appear wherever one space is permitted. Line numbers need only be used on statements to which control is passed. The line numbers do not have to be in ascending order. Using identifiers longer than two characters and indenting statements to enhance readability does not affect the size of the object file created by the

compiler.

The first statement of a source program may be used to specify certain compiler options. If present, this statement must begin with a dollar sign(\$) in column one and be followed by the letter or letters indicating the options which are desired. The letters may be separated by any number of blanks. Invalid letters or characters are ignored. At the end of the chapter there is a list of valid compiler options, and their initial settings. Toggle A is used for compiler debugging. Toggle B suppresses listing of the source program except for statements with errors. Toggle C compiles the program but does not create a INT file. Normally the BASIC-E compiler converts all letters appearing in identifiers or reserved words to uppercase. If Toggle D is set this conversion is not performed. Letters appearing in strings are never converted to uppercase. Toggle E causes code to be generated by the compiler so that, upon detection of a run-time error, the source statements line which was being executed at the time the error occurred is listed along with the error message.

The compiler begins execution by opening the source file specified as a parameter and compiles each BASIC-E statement producing an object file in the BASIC-E machine language with the same name as the source program but of type "INT". The source program may be listed on the output device with any error messages following each line of the program. There is a list of the compiler error messages at the end of the chapter. Errors 50, T0 and V0 indicate storage used by the compiler has been exceeded. These errors require recompilation of the program with more space allocated to the particular vector involved. If no errors occur during compilation, the object file may be executed by the run-time monitor.

The run-time monitor consists of two programs. The first program initializes the floating point package and then reads the intermediate language file specified as a parameter. As the INT file is read, the floating point constants, BASIC-E machine code, and the data area are built. At this time branch addresses and references to floating point constants and the program reference table (PRT) are relocated to reflect actual machine addresses. Control is then passed to the interpreter for execution of the BASIC-E machine language. The program which builds the machine resides in the free storage area and is overwritten as space is allocated by the interpreter. A list of the run-time error messages appears at the end of the chapter.

The BASIC-E machine is divided into a static and varying area. The static section consists of the BASIC-E Run Time Monitor, constants, machine code and data statements. The varying area, which includes all the remaining memory, stores program variables, the stack and the free storage area. The stack is a circular queue and therefore it is not possible to overflow the stack. However, if the stack wraps around onto itself the results could be meaningless. The size of the stack is initially set at 12 but may be changed by recompiling the interpreter. The free storage area is used to dynamically allocate arrays and strings. Storage is freed as soon as it is no longer required by the program.

All requests for input from the console are prompted with question mark. If more information is entered than was requested, or if insufficient information is entered, a warning occurs and the entire line of data must be entered again. A program may be terminated by typing a control-Z followed by a carriage return in response to an input request.

Disk files may be read, written or updated by the BASIC-E programmer using both sequential and random access. There are two basic types of files, those with no declared record size, which are referred to as unblocked, and those

with user-specified record size. The latter are blocked files. Blocked files may be processed either sequentially or randomly, while unblocked files must be accessed sequentially. All data in files is stored as ASCII characters. Either a comma or carriage return denotes the end of field. Blanks are ignored between fields.

In unblocked files there is no concept of a record as such. Each reference to the file either reads from or writes to the next field. At the end of each <write statement> a carriage return and a line feed are written to the file. This provides compatibility with the operating system utilities such as TYPE and also allows files to be created by the text editor.

Blocked files consist of a series of fixed length records. The user specifies the logical record length with the <file statement>. An error occurs if a line feed is encountered during a read from a blocked file or if the current record being built exceeds the block size during a write. At the end of a write statement any remaining area in the record is filled with blanks, and then a carriage return and linefeed are added.

The following features are not available in BASIC-E, but they are available in CBASIC --

1. PEEK or POKE
2. PRINT USING
3. CALL to machine code programs
4. LPRINT

The following are the BASIC-E commands:-

ABS (<expression>)

The ABS function returns the absolute value of the <expression>. The argument must evaluate to a floating point number.

eg:

ABS (X)

ABS (XY)

ASC (<expression>)

The ASC function returns the ASCII numeric value of the first character of the string <expression>. The argument must evaluate to a string. If the length of the string is 0 (null string) an error will occur.

eg:

ASC (AS)

ASC (^X')

ASC (RIGHTS(AS,7))

ATN (<expression>)

The ATN function returns the arctangent of the <expression>. The argument must evaluate to a floating point number. (All other inverse trigonometric functions may be computed from the arctangent using simple identities.)

eg:

ATN (X)

```
ATN (SQR(SIN(X)))
```

```
CHRS (<expression>)
```

The CHRS function returns a character string of length 1 consisting of the character whose ASCII equivalent is the <expression> converted to an integer modulo 128. The argument must evaluate to a floating point number.

eg:

```
CHRS (A)
```

```
CHRS (12)
```

```
CHRS ((A+B/C)*SIN(X))
```

CHRS can be used to send control characters such as a linefeed to the output device. The following statement would accomplish this:

```
PRINT CHR$(10)
```

```
CLOSE
```

```
[<line number>] CLOSE expression (,<expression>)
```

The CLOSE statement causes the file specified by each <expression> to be closed. Before the file may be referenced again it must be reopened using a FILE statement.

An error occurs if the specified file has not previously appeared in a FILE statement.

A CLOSE statement sometimes produces an apparently erroneous error message from the compiler. Usually the CLOSE statement can be omitted as all files are automatically closed at the end of program execution.

In some cases, when a file is written with a BASIC-E program, garbage is left in the file after the last record. This garbage shows up when the file is later TYPED, LISTed, or EDited. If you have trouble with this program, try putting an ASCII control-Z character:

```
PRINT $n; CHR$(26)
```

where n is the file number.

eg:

```
CLOSE 1
```

```
150 CLOSE I,K, L*M-N
```

(On normal completion of a program all open files are closed. If the program terminates abnormally it is possible that files created by the program will be lost.)

```
COS (<expression>)
```

COS is a function which returns the cosine of the <expression>. The argument must evaluate to a floating point number expressed in radians. A floating point overflow occurs if the absolute value of the <expression> is greater than two raised to the 24th power times pi radians.

eg:

```
COS (B)
```

```
COS (SQR(X-Y))
```

COSH (<expression>)

COSH is a function which returns the hyperbolic cosine of the <expression>. The argument must evaluate to a floating point number.

eg:

COSH (X)

COSH (X^2+Y^2)

DATA

[<line number>] DATA constant (,<constant>)

DATA statements define string and floating point constants which are assigned to variables using a READ statement. Any number of DATA statements may occur in a program. The constants are stored consecutively in a data area as they appear in the program and are not syntax checked by the compiler. Strings maybe enclosed in quotation marks or optionally delimited by commas.

eg:

10 DATA 10.0,11.72,100

DATA ^XYZ'',11.,THIS IS A STRING

DEF

[<line number>] DEF <function name> (<dummy argument list>) = <expression>

The DEF statement specifies a user defined function which returns a value of the same type as the<function name>. One or more <expressions> are passed to the function and used in evaluating the expression. The passed values may be in string or floating point form but must match the type of the corresponding dummy argument. Recursive calls are not permitted.

The <expression> in the define statement may reference <variables> other than the dummy arguments, in which case the current value of the <variable> is used in evaluating the <expression>. The type of the function must match the type of the <expression>.

eg:

10 DEF FNA(X,Y) = X + Y - A

DEF FNBS(AS,BS) = AS + BS + CS

DEF FN.COMPUTE (A,B) = A + B - FNA(A,B) + D

DEF FNXY.COMBINES(AS+BS)=AS+", "+BS

DIM

[line number] DIM <identifier> (<subscript list>) (<identifier> (<subscript list>))

The dimension statement dynamically allocates space for floating point or sting arrays. String array elements may be of any length up to 255 bytes and change in length dynamically as they assume different values. Initially, all floating point arrays are set to zero and all string arrays are null strings. An array must be dimensioned explicitly; no default options are provided. Arrays are stored in row major order.

Expressions in subscript lists are evaluated as floating point numbers and rounded to the nearest integer when determining the size of the array. All subscripts have an implied lower bound of 0. Any number of subscripts are

allowed. All arrays must be DIMensioned and the DIM statement should appear in the program before the array is referenced.

When array elements are reference a check is make to ensure the element resides in the reference array.

Eg:  
 DIM N(100)  
 DIM BIT(2,4,3,3,2,2,8), DATASTRING\$(10,5,23)  
 DIM A(10,20), B(10)  
 DIM BS(2,5,10),C(I + 7.3,N),D(I)  
 DIM X(A(I),M,N)

(A <DIM statement> is an executable statement, and each execution will allocate a new array.)

END  
 [line number] END

An END statement indicates the end of the source program. It is optional and, if present, it terminates reading of the source program. If any statements follow the END statement they are ignored.

eg:  
 10 END  
 END

FILE  
 [<line number>] FILE <variable>] FILE <variable> [((<expression>))  
 (,<variable>[((<expression>))])

Opens one or more files for use by the program. The order of the names determines the numbers used to reference the files in READ and PRINT statements. The value assigned to the first simple variable is file 1, the second is file 2, and so forth. There may be any number of FILE statement in a program, but there is a limit to the number of files which may be opened at one time. Currently this limit is set at 6 files.

The optional <expression> designates the logical record length of the file. If no length is specified, the file is written as a continuous string of fields with carriage return linefeed characters separating each record. If the record length is present, a carriage return linefeed will be appended to each record. The <variable> must not be subscripted and it must be of type string.

Eg:  
 FILE INPUT\$, OUTPUT\$  
 FILE TABLE.INCS, TAX.INCS(160), PAY.AMT.DAYS(N\*3-J)

(The run-time monitor will always assign the lowest available (not previously assigned) number to the file being opened. this if files are closed and others opened it is possible that number assignment may vary with program flow.)

FOR  
 [<line number>] FOR <index> = <expression> TO <expression> [STEP <expression>]

Execution of all statements between the FOR statement and its corresponding NEXT statement is repeated until the indexing variable, which is incremented by

the STEP <expression> after each iteration, reaches the exit criteria. If the step is positive, the loop exit criteria is that the index exceeds the value of the TO <expression>. If the step is negative, the index must be less than the TO <expression> for the exit criteria to be met.

The <index> must be an unsubscripted variable and is initially set to the value of the first <expression>. Both the TO and STEP expressions are evaluated on each loop, and all variables associated with the FOR statement may change within the loop. If the STEP clause is omitted, a default value of 1 is assumed. a FOR loop is always executed at least once. A step of 0 may be used to loop indefinitely.

Eg:  
 FOR I = 1 TO 10 STEP 3  
 FOR INDEX = J\*K-L TO 10\*SIN(X)  
 FOR I = 1 TO 2 STEP 0

(If a step of 1 is desired the step clause should be omitted The execution will be substantially faster since less run time checks must be made.)

FRE

The FRE function returns the number of bytes of unused space in the free storage area.

eg:  
 FRE

FN  
 FN<identifier>

Any <identifier> starting with FN refers to a user defined function. The <function name> must appear in a DEF statement prior to being used in an <expression>. There may not be any spaces between the FN and the <identifier>.

eg:  
 FNA  
 FN.BIGGER.\$

GOSUB  
 [<line number>] GOSUB <line number>  
 [<line number>] GO SUB<line number>

The address of the next sequential instruction is saved on the run-time stack, and control is transferred to the subroutine labelled with the <line number> following the GOSUB or GO SUB.

Eg:  
 10 GOSUB 300  
     GO SUB 100

( The max depth of GOSUB calls allowed is controlled by the size of the run-time stack which is currently set at 12.)

GOTO  
 [<line number>] GOTO <line number>  
 [<line number>] GO TO<line number>



Execution continues at the statement labelled with the <line number> following the GOTO or GO TO.

Eg:

```
100 GOTO 50
    GO TO 10
```

IF

```
[<line number>] IF <expression> THEN <line number>
[<line number>] IF <expression> THEN <statement list>
[<line number>] IF <expression> THEN <statement list> ELSE <statement list>
```

If the value of the <expression> is not 0 the statements which make up the <statement list> are executed. Otherwise the <statement list> following the ELSE is executed, if present, or the next sequential statement is executed.

In the first form of the statement if the <expression> is not equal to 0, an unconditional branch to the label occurs.

Eg:

```
IF A$ B$ THEN X=Y*Z
IF (A$<B$) AND (C OR D) THEN 300
IF B THEN X=D.0 : GOTO 200
IF J AND K THEN GOTO 11 ELSE GOTO 12
```

IF END

```
[<line number>] IF END £<expression> THEN <line number>
```

If during a read to the file specified by the <expression>, an end of file is detected control is transferred to the statement labelled with the line number following the THEN.

Eg:

```
IF END £ 1 THEN 100
10 IF END £ FILE.NUMBER - INDEX THEN 700
```

( On transfer to the line number following the THEN the stack is restored to the state prior to the execution of the READ statement which caused the end of file condition.)

INP

```
INP (<expression>)
```

The INP function performs an input operation on the 8080 machine port represented by the value of the <expression> modulo 256 returning the resulting value. The argument must evaluate to a floating point number.

Eg:

```
INP (2)
INP (CURRENT.INPUT.PORT)
```

INPUT

```
[<line number>] INPUT [<prompt string>;]
<variable> (<,>,<variable>)
```

The <prompt string>, if present, is printed on the console. A line of input data is read from the console and assigned to the variables as they appear in

the variable list. The data items are separated by commas and/or blanks and terminated by a carriage return. Strings may be enclosed in quotation marks. If a string is not enclosed by quotes, the first comma terminates the string. If more data is requested than was entered, or if insufficient data items is entered, a warning is printed on the console and the entire line must be reentered.

Eg:

```
10 INPUT A,B
    INPUT "SIZE OF ARRAY?";N
    INPUT "VALUES?";A(I),B(I),C(A(I))
```

( Trailing blanks in the <prompt string> are ignored. One blank is always supplied by the system.)

INT

INT (<expression>)

The INT function returns the largest integer less than or equal to the value of the <expression>. The argument must evaluate to a floating point number.

Eg:

```
INT (AMOUNT / 100)
INT (3 8 X 8 SIN(Y))
```

LEFTS

LEFTS (<expression>,<expression>)

The LEFTS function returns the n leftmost characters of the first <expression>, where n is equal to the integer portion of the second <expression>. An error occurs if n is negative. If n is greater than the length of the first <expression> then the entire expression is returned. The first argument must evaluate to a string and the second to a floating point number.

Eg:

```
LEFTS (A$,3)
LEFTS(CS*DS,I-J)
```

LEN

LEN (<expression>)

The LEN function returns the length of the string <expression> passed as an argument. Zero is returned if the argument is the null string.

Eg:

```
LEN (A$)
LEN(CS + BS)
LEN (LASTNAMES + "," + FIRSTNAMES)
```

LET

[<line number>] [LET] <variable> = <expression>

The <expression> is evaluated and assigned to the <variable> appearing on the left side of the equal sign. The type of the <expression>, either floating point or string, must match the type of the <variable>.

Eg:

```
100 LET A = B + C
      X(3,A) = 7.32 * Y = X(2,3)
73 W = (A<B) OR (CS>DS)
      AMOUNTS = DOLLARSS + "." + CENTSS
```

LOG

LOG (<expression>)

The LOG function returns the natural logarithm of the absolute value of the <expression>. The argument must evaluate to a non-zero floating point number.

Eg:

```
LOG (X)
LOG((A + B)/D)
LOG10 = LOG(X)/LOG(10)
```

MIDS

MIDS (<expression>,<expression>,<expression>)

The MIDS function returns a string consisting of the n characters of the first <expression> starting at the mth character. The value of m is equal to the integer portion of the second <expression> while n is the integer portion of the third <expression>.

The first argument must evaluate to a string, and the second and third arguments must be floating point numbers. If m is greater than the length of the first <expression> a null string is returned. If n is greater than the number of character are returned. An error occurs if m or n is negative.

Eg:

```
MIDS(AS,I,J)
MIDS(BS+CS,START,LENGTH)
```

ON

```
(1)\[<line number>] ON <expression> GOTO <line number> (, <line number>)
(2)\[<line number>] ON <expression> GO TO <line number> (, <line number>)
(3)\[<line number>] ON <expression> GOSUB <line number> (, <line number>)
(4)\[<line number>] ON <expression> GO SUB <line number> (, <line number>)
```

The <expression>, rounded to the nearest integer value, is used to select the <line number> at which execution will continue. If the <expression> evaluates to 1 the first <line number> is selected and so forth. In the case of an ON... GOSUB statement the address of the next instruction becomes the return address.

An error occurs if the <expression> after rounding is less than one or greater than the number of <line numbers> in the list.

Eg:

```
10 ON I GOTO 10, 20, 30, 40
      ON J*K-M GO SUB 10, 1, 1, 10
```

NEXT

[<line number>] NEXT [<identifier> (<identifier>)]

A NEXT statement denotes the end of the closest unmatched FOR statement. If the optional <identifier> is present it must match the index variable of the

FOR statement being terminated. The list of <identifiers> allows matching multiple FOR statements. The <line number> of a NEXT statement may appear in an ON or GOTO statement, which case execution of the FOR loop continues with the loop variables assuming their current values.

Eg:

```
10 NEXT
    NEXT I
    NEXT I, J, K
```

OUT

```
[<line number>] OUT <expression>, <expression>
```

The low-order eight bits of the integer portion of the second <expression> is sent to the 8080 machine output port selected by the integer portion of the first expression modulo 256. Both arguments must evaluate to floating point numbers.

Eg:

```
100 OUT 3,10
    OUT PORT.NUM, NEXT.CHAR
```

PRINT

```
(1)\[<line number>] PRINT £<expression>,<expression>; <expression>
(<expression>)
(2)\[<line number>] PRINT £<expression>; <expression> (<expression>)
(3)\[<line number>] PRINT <expression><delim>(<expression><delim>)
```

A PRINT statement sends the value of the expressions in the expression list to either a disk file (type(1) and (2) or the console (type (3)). A type (1) PRINT statement sends a random record specified by the second <expression> to the disk file specified by the first <expression>. An error occurs if there is insufficient space in the record for all values.

A type (2) PRINT statement outputs the next sequential record to the file specified by the <expression> following the £.

A type (3) PRINT statement outputs the value of each <expression> to the console. A space is appended to all numeric values and if the numeric item exceeds the right margin then the print buffer is dumped before the item is printed. The <delim> between the <expressions> may be either a comma or a semicolon. The comma causes automatic spacing to the next tab position (14,28,42,56). If the current print position is greater than 56 then the print buffer is printed and the print position is set to zero. A semicolon indicates no spacing between the printed values. if the last <expression> is not followed by a <delim> the print buffer is dumped and the print position set equal to zero. The buffer is automatically printed anytime the print position exceeds 71. PRINT sometimes makes an error of one in the seventh significant digit; thus, for example, 93 may print out as 92.99999. This error occurs on printout and does not effect arithmetic done in the program.

Eg:

```
100 PRINT £1;A,B,AS+"*"
    PRINT £ FILE, WHERE; A/B,D,"END"
    PRINT A, B, "THE ANSWER IS";x
```

## RANDOMIZE

```
[<line number>] RANDOMIZE
```

A RANDOMIZE statement initializes the random number generator. The RANDOMIZE statement is effective only if an INPUT statement has been executed earlier in the program. The variable time it takes the user to respond at the terminal is used to randomize the random number generator.

Eg:

```
10 RANDOMIZE
   RANDOMIZE
```

## READ

```
(1)\[<line number>] READ <expression>,<expression>;<variable>), <variable>)
(2)\[<line number>] READ £ <expression>;<variable>(<variable>), <variable>)
(3)\[<line number>] READ £<variable>),<variable>)
```

A READ statement assigns values to variables in the variable list from either a file (type (2) and (3)) or from a DATA statement (type (1)). Type (2) reads a random record specified by the second expression from the disk file specified by the first expression and assigns the fields in the record to the variables in the variable list. Fields may be floating point or string constants and are delimited by a blank or comma. Strings may optionally be enclosed in quotes. An error occurs if there are more variables than fields in the record.

The type (3) READ statement reads the next sequential record from the file specified by the expression and assigns the fields to variables as described above.

A type (2) READ statement assigns values from DATA statements to the variables in the list. DATA statements are processed sequentially as they appear in the program. An attempt to read past the end of the last data statement produces an error.

Eg:

```
100 READ A,B,CS
200 READ £ 1,I; PAY.REG,PAY.OT,HOURS.REG,HOURS.OT
   READ £ FILE.NO; NAMES,ADDRESS$,PHONES$,ZIP
```

## REM

```
[<line number>] REM (<remark>)
[<line number>] REMARK (<remark>)
```

A REM statement is ignored by the compiler and compilation continues with the statement following the next carriage return. The REM statement may be used to document a program. REM statements do not affect the size of programs that may be compiled or executed. An unlabelled REM statement may follow any statement on the same line. And the <line number> may occur in a GOTO GOSUB, or On statement.

Eg:

```
10 REM THIS IS A REMARK
   REMARK THIS IS ALSO A REMARK
   LET X = 0 REM INITIAL VALUE OF X
```

RESTORE  
[<line number>] RESTORE

A RESTORE statement repositions the pointer into the data area so that the next value read with a READ statement will be the first item in the first DATA statement. The effect of a RESTORE statement is to allow rereading the DATA statements.

Eg:  
RESTORE  
10 RESTORE

RETURN  
[<line number>] RETURN

Control is returned from a subroutine to the calling routine. The return address is maintained on the top of the run-time monitor stack. No check is made to insure that the RETURN follows a GOSUB statement.

Eg:  
130 RETURN  
    RETURN

RIGHT\$  
RIGHT\$ (<expression>,<expression>)

The RIGHT\$ function returns the n rightmost characters of the first <expression>. The value of n is equal to the integer portion of the second <expression>. If n is negative an error occurs; if n is greater than the length of the first <expression> then the entire <expression> is returned. The first argument must produce a string and the second must produce a floating point number

Eg:  
RIGHT\$(XS,1)  
RIGHT\$(NAMES,LNG.LAST)

RND

The RND function generates a uniformly distributed random number between 0 and 1.

Eg:  
RND

SGN  
SGN (<expression>)

The SGN function returns 1 if the value of the <expression> is greater than 0, -1 if the value is less than 0 and 0 if the value of the <expression> is 0. The argument must evaluate to a floating point number.

Eg:

```
SGN(X)
SGN(Z - B + C)
```

```
SIN
SIN (<expression>)
```

SIN is a predefined function which returns the sine of the <expression>. The argument must evaluate to a floating point number in radians.

A floating point overflow occurs if the absolute value of the <expression> is greater than two raised to the 24th power times pi.

```
Eg:
X = SIN(Y)
SIN(A - B/C)
```

```
SINH
SINH (<expression>)
```

SINH is a function which returns the hyperbolic sine of the <expression>. The argument must evaluate to a floating pointer number.

```
Eg:
SINH(Y)
SINH(B<C)
```

```
STR$
STR$ (<expression>)
```

This STR\$ function returns the ASCII string which represents the value of the <expression>. The argument must evaluate to a floating point number.

```
Eg:
STR$(X)
STR$(3.141617)
```

```
SQR
SQR (<expression>)
```

SQR returns the square root of the absolute value of the <expression>. The argument must evaluate to a floating point number.

```
Eg:
SQR (Y)
SQR(X^2 + Y^2)
```

```
STOP
[<line number>] STOP
```

Upon encountering a <STOP statement> program execution terminates and all open files are closed. The print buffer is emptied and control returns to the host system. Any number of STOP statements may appear in a program.

A STOP statement is appended to all programs by the compiler.

```
Eg:
10 STOP
STOP
```

TAB  
TAB (<expression>)

The TAB function positions the output buffer pointer to the position specified by the integer value of the <expression> rounded to the nearest integer modulo 73. If the value of the rounded expression is less than or equal to the current print position, the print buffer is dumped and the buffer pointer is set as described above.

The TAB function may occur only in PRINT statements.

Eg:  
TAB (10)  
TAB (I + i)

TAN  
TAN (<expression>)

TAN is a function which returns the tangent of the expression. The argument must be in radians.

An error occurs if the <expression> is a multiple of  $\pi/2$  radians.

Eg:  
10 TAN(A)  
TAN(X - 3\*COS(Y))

VAL  
VAL (<expression>)

The VAL function converts the number in ASCII passed as a parameter into a floating point number. The <expression> must evaluate to a string.

Conversion continues until a character is encountered that is not part of a valid number or until the end of the string is encountered.

Eg:  
VAL(AS)  
VAL("3.789" + "E-07" + "THIS IS IGNORED")

#### COMPILER ERROR MESSAGES

CE	Could not close file.
DE	Disk error
DF	Could not create INT file; disk or directory is full.
DL	Duplicate labels or synchronization error.
DP	Identifier in DIM statement previously defined.
FC	Identifier in FILE statement previously defined.
FD	Predefined function name previously defined.
FI	FOR loop index is not a simple floating point variable.
FN	Incorrect number of parameters in function reference.
FP	Invalid parameter type in function reference.
FU	Function is undefined.
IC	Invalid character in BASIC statement.
IE	Expression in IF statement is not of type floating point.
IS	Subscripted variable not previously dimensioned.
IU	Array name used as simple variable.



## Appendix B - Introduction to BASIC-E

MF	Expression is of type string where only floating point is allowed.
MM	Expression contains string and floating point variables in mixed mode expression.
NI	Identifier following NEXT does not match FOR statement index.
NP	No applicable production exists.
NS	No BAS file found
NU	NEXT statement without corresponding FOR statement.
SN	Incorrect number of subscripts.
SO	Compiler stack overflow
TO	Symbol table overflow.
UL	Undefined label.
VO	VARC overflow.

### RUN-TIME MONITOR ERROR MESSAGES

AC	Null string passed as parameter to ASC function.
CE	Error closing a file.
DR	Disk read error (reading unwritten data in random access).
DW	Error writing to a file.
DZ	Division by zero.
EF	EOF on disk file; no action specified
ER	Exceeded record size on block file.
II	Invalid record number in random access.
FU	Accessing an unopened file.
ME	Error attempting to create a file.
MF	File identifier too large or zero.
NE	Attempt to raise a number to a negative power.
NI	No INT file found in directory.
OD	Attempt to read past end of data area.
OE	Error attempting to open a file.
OI	Index in ON statement out of bounds.
RE	Attempt to read past end of record on blocked file.
RU	Unblocked file used with random access.d
SB	Array subscript out of bounds.
SL	String length exceeds 255.
SS	Second parameter of MIDS is negative.
TZ	Attempt to evaluate tangent of pi over two.

### Compiler options:-

- A List productions (for compiler debugging).(o)
- B List only source statements with errors. (o)
- C Do not create INT file; syntax check only.(o)
- D Convert lowercase to uppercase (n)
- E Generate line number code (o)

## Appendix C – The Implementation

The Amstrad CP/M+ caters for one or two floppy disc drives. Where there is only one drive, drive B becomes a logical drive to allow disc to disc copying and editing using standard CP/M utilities. Whenever a new logical drive is accessed, the user is prompted to change the disc. The PCW 8256 also has an area of memory that emulates a disc drive (RAM disc)

The TPA is usually 61K. The console emulates the main VT52 and Zenith screen codes, and is incompatible with the CP/M 2.2 implementation. Disc errors actuate BIOS error messages over and above the standard CP/M+ ones, allowing the correction of remediable disc errors.

There is an extended jumpblock accessed via the BIOS jump vector 'USERF'. The SIO driver is implemented where this optional extra is fitted, but there is no support for the second SIO channel.

The PCW 8256 emulates the main EPSON FX-80 printer control codes in software for the built-in printer. It also has an 8-bit centronics interface support if the hardware is fitted. It is not possible to support other devices, such as large-capacity discs or high-speed VDU emulation, on the CP/M+, due to the fact that the implementation is not supplied in '.REL' file format with GENCPM.COM and GENCPM.DAT. The scope for patching is strictly limited.

There are three physical character-oriented devices; the CRT (screen and keyboard ), LPT (the latched Centronics 7 bit parallel interface for the printer) and the SIO (RS232 serial interface if fitted). CRT and SIO are both input and output devices but the LPT is output only. The second SIO channel is not supported. The PCW 8256 has an extra device, CEN, for the optional centronics interface.

### THE CRT

The CRT passes through an emulator before using the firmware. This is generally compatible with the VT52 and Zenith and makes the installation of standard application programs more simple.

The emulator does not precisely emulate any particular terminal but is similar to many. It is almost the same as that implemented by Digital Research for their 16 bit operating systems.

#### Characters.

The emulator supports the full 8-bit character set including graphics  
Some control codes (00H...1FH) are interpreted :

BEL	(07H)	Sounds a bleep
BS	(08H)	Backspace. Moves the cursor left one column. If already at the left hand edge, and if wrapping is enabled then the cursor moves up one line to the right most column. If already in the first column of the top line do nothing.
LF	(0AH)	Line feed. Move the cursor down one line, scroll up if necessary.
CR	(0DH)	Carriage return. Move the cursor to column 0.

ESC (1BH) Escape. This is the header character to show the emulator that the next character is not to be taken literally but is a command to the emulator. These commands are as follows:-

- 0 Disable the status line.
- 1 Enable the status line. The bottom line of the screen is reserved for use by the BIOS as a status line. On it are displayed disc error messages and, on single drive systems, instructions to change the disc and whether the drive is A: or B:.. The status line can be disabled in which case the CRT device may use the bottom line and BIOS messages are displayed on the CONOUT: device.
- 2 Change the character set (language). It requires a following parameter in the range 0..7. The screen character set is then altered to a particular national character set by swapping character matrices within the normal ASCII range (£00..£7F) with characters in the extended range (£A0..£FF).
- 3 Set screen mode. Takes one further parameter, the screen mode in the range 0 to 2 The screen is cleared but the cursor position is not affected. All three screen modes are supported on the CPC 6128, none are supported on the PCW 8256 and the command is ignored.

Mode 0 25 rows, 20 columns

Mode 1 25 rows, 40 columns

Mode 2 25 rows, 80 columns

it is unlikely that any other than mode 2 will be required.

- A Cursor up. If on top row do nothing.
- B Cursor down. If on bottom row do nothing.
- C Cursor forward. If on rightmost column do nothing.
- D Cursor backwards. If on leftmost column do nothing.
- E Erase page. The cursor position is unaffected.
- H Home cursor. Moves to top left corner.
- I Reverse index. Move up one row. Scroll down if necessary.
- J Erase to end of page. Includes character at cursor position. The cursor position is unaffected.
- K Erase to end of line. Includes character at cursor position. The cursor position is unaffected.
- L Insert line. All rows below and including the cursor row are scrolled down. The cursor row is cleared. The cursor position is unaffected.

- M Delete line. All rows below and including the cursor row are scrolled up. The bottom row is cleared. The cursor position is unaffected.
  
- N Delete character. All characters to the right of the cursor are shuffled left one character position. The character at the end of the row is cleared. The cursor position is unaffected.
  
- X (PCW 8256 only) Set text viewport. Takes four parameters: top row of viewport + £20, left column of viewport + £20, height of viewport in rows - 1 + £20, width of viewport in columns - 1 + £20. If necessary the cursor is moved within the viewport. The terminal emulator in the PCW 8256 restricts its operation to within a viewport. The viewport may be all, or part, of the screen aligned on character (8 pixel) boundaries. The size and shape of the viewport is changed by the ESC X sequence or by selecting 24 x 80 mode or by enabling or disabling the status line. There is only one viewport. Character cells to the left and right of the viewport are affected by scrolling the viewport. Character cells above and below the viewport are not affected.
  
- Y Cursor position. Takes two further parameters, first is row number + £20, second is column number + £20. Moves cursor to given position, if position is beyond the edge of the screen the cursor is moved to the edge of the screen.
  
- b Set foreground colour. (irrelevant on the PCW 8256 ) The character following specifies the colour. The lower six bits are treated as three 2 bit numbers each specifying the intensity of one of the three primary colours. Bits 0,1 for blue, bits 2,3 for red and bits 4,5 for green. The 3 levels of intensity are mapped as follows:
 

Colour parameter : 0 1 2 3

CPC6128 : 0 1 1 2

This command affects all characters on the screen.
  
- c Set background colour. (irrelevant on the PCW 8256 ) The parameter following specifies the colour and is treated in the same way. The command affects all characters on the screen.
  
- d Erase to the beginning of page. Includes character at cursor position. The cursor position is unaffected.
  
- e Enable cursor blob.
  
- f Disable cursor blob.
  
- j Save the cursor position.
  
- k Restore the cursor position as saved by ESC j.
  
- l Erase line. The cursor position is unaffected.

- o Erase beginning of line. Includes character at cursor position. The cursor position is unaffected.
- p Enter reverse video mode.
- q Exit reverse video mode.
- r Enter underline mode (not supported on CPC6128)
- u Exit underline mode (not supported on CPC6128)
- v Wrap at end of line.
- w Discard at end of line.
- x Enter 24 x 80 mode, selects screen mode 2 if a CPC 6128. In the case of the PCW 8256, 50Hz machines have a screen size of 32 lines by 90 columns; 60Hz machines have a screen size of 25 lines by 90 columns. The bottom row may be used as a status line. Some application programs may require a "standard" 24 x 80 screen. Sending ESC x will enable 24 x 80 mode regardless of the actual size of the screen. If the cursor is outside the 24 x 80 area it is moved inside the area at the nearest edge. ESC y will restore the screen to its full size, this depends on the machine, the country and whether or not the status line is enabled. Both ESC x and ESC y will clear the entire screen.
- y Exit 24 x 80 mode, selects screen mode 2. This is irrelevant to the 6128 but is put in for reasons of compatibility.

If any other character follows an ESC, it is displayed and the cursor advanced as usual. This allows the display of certain control codes that would otherwise be acted on.

To prevent unsightly cursor flashing whilst outputting text to the screen the cursor is not turned on until 1/10 second after the last character was written.

#### THE SIO

The physical device 'SIO' drives channel A of a Zilog SIO or DART (if fitted as an optional extra). The serial port should either be the Amstrad serial port or one that emulates it exactly. The presence of the serial interface is detected during cold boot and displayed in the initial signon message. If found the serial interface becomes device number 2 "SIO" and the AUX: device is initially redirected to it though this assignment can be changed using the DEVICE utility. Otherwise the AUX: device is redirected to the null device and characters are 'thrown away'. The second serial channel is not supported.

The parity, number of stop bits, number of receive and transmit data bits, and baud rates can all be set from the software utilities. Baud rates have a certain error which is unusually large due to hardware limitations:-

50	0.10%
75	0.12%
110	0.13%
134.5	-0.14%
150	-0.14%
300	0.18%
600	-0.26%
1200	-0.26%
1800	-0.74%
2400	-0.26%
3600	0.18%
4800	-0.26%
7200	-2.18%
9600	-0.26%
19200	7.62%

The inaccuracy is particularly accentuated with 7200 and 19200 baud and use of these baud rates is not recommended.

The SIO can optionally use software or hardware handshaking. If hardware handshaking is in force, RTS is always high. (This will cause difficulties with half-duplex modems as when there is no character DTR is raised, it is dropped when a character is read). Characters are not sent until CTS goes high and the SIO "all sent" condition. When hardware handshaking is disabled, RTS and DTR are always high, and character throughput depends merely on the status of either the transmit or receive buffer. In either case, there is a ten-second timeout. After this timeout, the BIOS message

"DEVICE not ready - Retry, Ignore or Cancel?"

appears on the screen. If the problem can be corrected, then do so and type r, whereas typing I or C returns to CP/M without transmitting the character. C throws all further characters away by directing them to a dummy device called NUL.

#### The LPT device

Whereas the LPT device is a standard 7 bit parallel port on the CPC 6128, on the PCW 8256 it supports a 'virtual Epson FX-80' emulation on the actual dumb Seikosha mechanism. The actual emulation is done by the Z80. We will refer to this emulation as the 'printer'. The printer is a dot matrix printer capable of printing both characters and graphics, with the following options :

- High or Draft Quality characters
- 10, 12, 17.14 characters per inch, or proportionally spaced characters
- Italic
- Double width characters
- Underlining
- Emboldening
- Double strike
- Superscripts and subscripts
- 6 or 8 lines per inch (and many others)
- 60 or 120 dots per inch graphics
- ASCII character set with eight national variants

The paper handling capabilities of the printer are :

- Auto load (Bail bar) action
- Paper between 52 and 76 gsm plus 2 copies each up to 40 gsm
- Paper up to 10 inches wide
- Print area 8.13 inches wide (8 and 2/15 inches)
- Continuous (tractor fed) and single sheet stationery
- Settable form length
- Settable perforation gap length
- Settable left and right margins
- Paper out detection, with paper out defeat
- Automatic printer pause between sheets with single sheet stationery

The printer may be controlled by programs sending escape sequences, or directly by the operator using the PTR key (the printer key). The escape sequences are designed so that programs equipped to drive an Epson FX-80 should be capable of driving the PCW8256's printer.

Pressing the PTR key puts the machine into Printer Control State. This halts the printer as soon as it has completed its current operation (printing a line or feeding paper). While in printer control the bottom line of the screen is taken over and used to show the state of the printer, and to provide a number of "buttons" which the operator may use to control the printer. These software "buttons" provide the functions usually given by real buttons on other printers. Pressing EXIT causes the machine to leave printer control, and return to whatever it was doing.

Loading paper automatically enters printer control, which halts the printer to allow the operator to ensure that the paper is correctly loaded before printing continues.

The PAPER utility is provided to facilitate the settings for form length, gap length, single sheet and so on. The PAPER utility may be used in PROFILE.SUB to set these values when CP/M is loaded.

Many printers have switches which set the values of some of the printer's parameters when it is turned on or reset. To provide the same function the PCW8256 printer software maintains default settings for almost all printer parameters. A special escape sequence is provided to copy the current printer settings to the default settings.

#### The Character Printing Capabilities.

The PCW8256 printer has an extensive set of character printing capabilities. This section describes them and the ways in which the various options interact. To help in this description some of the printer mechanism is also described.

The printer has a nine needle head, with the needles 1/72nd of an inch apart. When printing the head moves at one of two speeds, where the fast speed is approximately 8 inches per second and the slow speed is half that. The head can be moved and printing can be done in either direction. Once the head is moving each needle may be fired once every 1,745 micro-seconds. When printing in fast speed this timing means that dots on the paper are separated horizontally by a gap approximately the same size as the dot. In slow speed the dots join together. The unit of time used in the control of the needles is 174.5 micro-seconds, which corresponds to a 1/720nd of an inch across the paper in fast speed and a 1/1440th of an inch in slow speed. When feeding paper the

unit of movement is a 1/432nd of an inch, which gives six steps to move down the distance between needles.

When printing in draft quality the character matrices are twelve dots wide by nine deep. Characters are printed in one pass at the fast speed, except for 17.14 pitch which is done at slow speed. When printing in high quality the character matrices have four times as many dots, twenty four wide by eighteen deep. Characters are printed in two passes at slow speed. The slow speed doubles the number of dots printed horizontally. The second pass doubles the number of dots printed vertically by performing a 3 step vertical move between the two passes, to position the second set of dots half way between the first. To give the most accurate registration possible between the two passes they are both performed in the same direction.

The difference between 10 and 12 character per inch printing is simply that the distance between dots is reduced by 1/6th to give 12 pitch. Proportionally spaced printing is basically 12 pitch, with some blank columns in the matrices removed. The 17.14 characters per inch printing (sometimes known as condensed printing) is done at slow speed. Double width printing is done by doubling the space occupied by each dot in the character matrix and filling in the space between.

Superscripts and subscripts are printed as half height characters using the draft matrices and printing in two passes with a 3 step vertical move between the passes (much like the high quality printing).

Emboldening is done by printing an extra dot to the right of each one in the matrix (unless there is a dot there already). This gives a "shadow" effect. All bold printing is done at slow speed.

Double strike is done by printing the same matrices in two passes with a 3 step vertical move between the each pass. This fills in the vertical gap between dots, and gives a darker image.

Italic printing is done by massaging the matrices to make them lean over. All italic printing is done at slow speed and is always done forwards.

There are potentially 384 different combinations of character printing options (not counting underline). Some combinations are, however, not possible, as follows :

- a. Double strike is not possible with high quality printing. This is because both require two passes. If high quality double strike is requested high quality bold is used (except in 17.14 pitch, where neither is possible, and except for superscripts or subscripts, where high quality is not possible).
- b. Neither high quality nor double strike printing are possible with superscripts or subscripts. This is because superscript and subscript also require two passes. All superscripts and subscripts are printed using draft matrices. If double strike superscripts or subscripts are requested they are printed emboldened (except in 17.14 pitch, where bold is not possible).
- c. Neither high quality nor bold printing are possible in 17.14 pitch. In both cases draft matrices are used with double strike (except in superscripts and subscripts, where double strike is not possible).



These restrictions and substitutions are performed automatically, reducing the total number of printable combinations to a mere 160.

#### Printer Paper Handling.

As far as the printer is concerned paper has the following properties:

a. Continuous or Single Sheet

The printer expects the tractor feed to be used with continuous stationery. With single sheet stationery the tractor feed is not used and the printer software makes a small adjustment to paper feeds to take into account the slight difference between the tractor and friction feed.

When single sheet stationery is in use the printer automatically detects feeds beyond the end of a page (or into the gap area), and halts the printer to wait for more paper to be loaded. The way in which single sheet stationery is handled means that it is not possible to print on the first inch of the paper. The printer software takes this into account when Top of Form is set, and sets the position to be one inch down the paper.

b. Form Length

For continuous stationery this is the distance between perforations.

For single sheet stationery this is the length of the paper in use.

c. Gap Length

This defines an area in which nothing is to be printed. The form length gives the total length of a piece of paper. Form length minus the gap length gives the part of the paper which may be printed on.

For continuous stationery the gap length may be set to avoid printing on or near the perforations.

For single sheet stationery the gap length may be set to avoid printing too near to the bottom of the paper.

The difference between the handling of continuous and single sheet stationery is quite marked, and the operator should take care to ensure that the printer is set up appropriately.

The printer has a paper sensor which tells it when paper is loaded, provided the paper is reasonably pale and is not positioned away from the sensor (far to the right for example). This is fine for most continuous stationery, but can be fooled by some single sheet stationery, particularly if the paper is preprinted. The paper sensor also detects no paper a little early on single sheet stationery. With single sheet the printer software detects the end of paper simply by keeping track of how far it has fed. The printer has a "Paper Out Defeat" option which, when set, causes it to disregard the paper sensor. Use of paper out defeat is recommended with single sheet stationery.

### Printer Control State.

Entering printer control state halts the printer as soon as it has completed its current operation (printing a line or feeding paper). If, for example, paper jams in the printer then pressing the PTR key will halt printing to allow the operator to deal with the problem. At other times the machine automatically enters printer control state to ensure that the printer is halted until the operator is satisfied that printing can continue.

In printer control state the bottom line of the screen is taken over to show the state of the printer and to give a number of "buttons". Pressing EXIT causes the machine to leave printer control state.

The printer control line has seven or eight fields. There is a cursor which may be moved from field to field using the left and right cursor keys. The first field gives the state of the printer, and may contain one of the following :

#### Active

The printer is currently doing something.

#### Online

The printer is halted, but will print when printer control state is left (when there is anything to be printed).

#### Offline

The printer is permanently halted. The operator may set the printer offline when it is online by pressing [-] (clear) when the cursor is on this field. The printer is set online by pressing [+] (set) when the cursor is on this field and the printer is offline.

#### Bail bar out

The operator has turned the auto load lever anti-clockwise or has otherwise pulled back the bail bar. If no paper can be detected this causes the printer to auto load paper (during the auto load the printer is Active). Entering bail bar out state also causes entry to printer control state.

#### Out of paper

The paper sensor cannot detect paper and paper out defeat is not set. This state will be cleared when the paper sensor can detect paper or paper out defeat is set. Note that if paper is detected then printer control state is entered automatically (unless the printer is offline).

#### Waiting for paper

When the printer software detects a move to or beyond the end of a page of single sheet stationery, it automatically puts the printer into this state. The state is cleared by auto load, or by pressing [-] (clear) when the cursor is on this field.

ERROR - underrun  
ERROR - printer RAM test  
ERROR - illegal command  
ERROR - print error

These are all errors detected by the printer hardware, and should not occur. RESET will clear the error, unless it is a printer RAM test error, which cannot be cleared.

#### No printer

The printer is not connected. RESET will cause the software to reconsider.

When the printer is in Online or Offline states the next three fields contain :

at line: nn or Top of Form

This gives the current position on the paper in "standard lines" (six lines per inch). Note that the first line is line 1. Note also that Top of Form on single sheet stationery is one inch down from the top of the paper, at line 7.

Pressing [+] (set) when on this field will set Top of Form.

#### LF

Pressing [+] (set) when on this field will feed 1 standard line (1/6th of an inch).

#### FF

Pressing [+] (set) when on this field will cause the printer to feed to the top of the next form.

The next three fields allow the operator to set or clear the following printer parameters:

Draft quality or High quality

[+] (set) or [-] (clear) when on this field will swap between draft and high quality printing. Note that changing the print quality will not affect any text in the printer buffer.

PO defeat:On or PO defeat:Off

[+] (set) will set Paper Out Defeat on. [-] (clear) will set it off. This takes immediate effect, and may change the printer state.

Hex:On or Hex:Off

[+] (set) will set Hexadecimal Dump On, [-] (clear) will set it off. In hexadecimal dump mode the printer does not print the characters or obey the control codes sent to it, but prints the hexadecimal value of each one. The output is always in 10 pitch across the complete width of the printer, giving twenty hexadecimal two digit numbers separated by spaces across the complete width of the printer. This

is irrespective of any margin or pitch or other settings. Note that changing the hexadecimal dump mode will not affect any text in the printer buffer.

#### RESET

Pressing [+] (set) when the cursor is over this field will reset the printer and discard all buffered text. All printer parameters are reset to their default values. If the printer is active RESET will abandon the current operation.

Printer control state is entered when the PTR key is pressed and automatically when :

The Bail bar is pulled back.

Paper is detected when it was previously out and the printer is online.

An attempt is made to print when in any of the following states:

- Bail bar out
- Waiting for paper
- Out of paper

The automatic entry to printer control state allows the operator to insert paper in the printer, or ensure that paper has been inserted correctly, before printing starts (or continues).

#### Auto Load - Bail bar.

The Auto Load sequence is started by turning the auto load lever anti-clockwise or otherwise pulling back the bail bar. If no paper can be detected the printer feeds paper until either paper is detected or until it has fed 2 and 3/8ths inches. If paper is detected the printer feeds a further 1 and 5/8ths inches. The intention is to feed the paper so that the top comes just under the bail bar when it is put forward again, which places the first line of print one inch down from the top of the paper. This positioning is less accurate if paper is not detected.

Auto load always clears Waiting for paper state, if the printer is in it.

Auto load sets Top of Form, unless paper was detected before any feed was done, in which case it has no effect on the position.

Auto load automatically causes entry to printer control state.

### RAM Disc

Drive M: is a RAM disc. That is an area of RAM which behaves just like a disc with the exception that all data is lost when the machine is reset or turned off. The size of RAM disc is determined during cold boot. On the standard 256K PCW8256 the RAM disc is 112K. Any contiguous add-on memory is used for the RAM disc.

### Disc driver

The disc driver supports one or two floppy disc drives. Two disc formats are supported: "system format" and "data only format" as per CP/M 2.2 on other AMSTRAD machines. The PCW 8256 has one extra format, potentially a whole family of them. (The precise details of the format are recorded on the boot sector as the "disc specification"). The system is easily patched to support the IBM PC CP/M single-sided format.

Only the boot sector is used from the system tracks on a system format disc. The remainder of these tracks are not used. The "system format" disc format is only necessary on the system disc that is used when the machine is initially "cold booted". All other discs can be of "Data only format".

Many of the disc driver routines are available to an application program via the extended BIOS jumpblock; for example the DISCKIT3 utility program uses these routines since there is no standard CP/M 3 facility for formatting discs.

### Single drive systems

On single drive systems two "logical" drives (A: and B:) are both mapped onto the one physical drive. The number of physical drives fitted is detected during cold boot.

The BIOS routine SELDSK deals with the logical to physical drive mapping. Whenever it is called it checks that the logical drive being selected is currently mapped onto the physical drive. If not a BIOS message is displayed to prompt the user to change the disc. This allows the use of standard utilities such as PIP to be used when there is only one physical drive.

The current assignment of logical to physical is displayed on the status line as "Drive is A:" or "Drive is B:".

The disc driver routine interfaces refer to the drives as "unit 0" and "unit 1". This terminology is introduced for reasons of upwards compatibility with other products which may have to make a clear distinction between physical drives (units) and logical drives (drives A: B: C: ... P:).

There are facilities for patching the BIOS for a range of discs. This is especially useful when using 5 1/4 in. drives and wishing to read the formats of other micros. In order to do this, an understanding of the disc drivers is necessary. This is given in the following paragraphs.

### Logical tracks and sectors

The disc driver routines require "logical" tracks and sectors. These are used to hide information concerning the number of sides and the actual sector numbers from CP/M 3, which knows nothing about them.

Logical track numbers on a single sided disc are the same as physical track numbers.

For double sided discs two options are available:

#### Flip sides

side 0 track 0 = logical track 0	
side 1 track 0	1
side 0 track 1	2
side 1 track 1	3
...	...

#### Up and over

side 0 track 0 = logical track 0	
side 0 track 1	1
...	...
side 0 last track	40
side 1 last track	41
side 1 last track - 1	42
...	...
side 1 track 0	79

Logical sectors hide the actual physical sector numbers. Logical sector numbers always start from 0.

Logical sector = physical sector - first sector

BIOS disc error messages are in terms of logical track and sector numbers. All this information is held in an eXtended Disc Parameter Block (XDPB).

### Disc Specification of the PCW 8256 format

The PCW8256 disc format is, in fact, a family of formats, the precise member of which is defined in the "disc specification" which is recorded on bytes 0..15 of sector 1, track 0 side 0. This enables the user to define special formats for special purposes or special devices. As the specification is read off the disc there is no need for any special patching of the XDPB for the PCW 8256.

Byte 0:	disc type
	0 means standard PCW8256 single sided format
	all other values reserved
Byte 1:	sidedness
	0 means single sided
	1 means double sided, flip sides
	2 means double sided, up and over
Byte 2:	number of tracks per side
Byte 3:	numbers of sectors per track
Byte 4:	log2 (sector size) - 7
Byte 5:	number of reserved tracks
Byte 6:	log2 (Block size) - 7
Byte 7:	number of directory blocks
Byte 8:	gap length (read/write)

# The Implementation of CP/M Plus on the 6128 and 8256

Byte 9: gap length (format)

Bytes 10..15: reserved

When a disc is logged on the disc specification is used to initialize the relevant XDPB.

### Extended Disc Parameter Blocks (XDPB)

Associated with each (logical) drive is an extended disc parameter block. This contains a standard DPB as required by CP/M 3 and information required by the BIOS to support the different formats. It may be patched in order to use different format discs provided that the restrictions detailed below are obeyed.

#### XDPB structure:

```

bytes 0..16 : standard CP/M 3 DPB.
byte 17    : sidedness
              0 means single sided
              1 means double sided flip sides
              2 means double sided up and over
byte 18    : number of tracks per side.
byte 19    : number of sectors per track
byte 20    : first sector number
byte 21,22 : sector size
byte 23    : gap length (read/write).
byte 24    : gap length (format).
byte 25    : bit 7 multi-track operation
              1 means multi-track operation
              0 means single track
              bit 6 modulation mode
              1 means MFM mode
              0 means FM mode
              bit 5 skip deleted data address mark
              1 means skip deleted data address mark
              0 means don't skip deleted address mark
              bits 4..0 = 0
byte 26    : freeze flag
              £00 means auto-detect disc format
              £FF means don't auto-detect disc format
    
```

Byte 25 is normally set to £60. Multi-track operation is not recommended.

Setting the freeze flag (byte 26) prevents the BIOS from trying to determine the format of a disc. This should be used when patching an XDPB for a non-standard format.

To find the XDPB for a particular drive use BDOS function 31.

The restriction on patching an XDPB is that the resulting disc structure must lie within the following maximum sizes:

```

Maximum 2 bit allocation vector = 91 bytes
Maximum checksum vector         = 32 bytes
Maximum hash table size         = 512 bytes
Maximum sector size             = 512 bytes
    
```

This corresponds to a disc of 160 tracks, 9 sectors per track, 512 bytes per sector, 1 reserved track, 128 directory entries, 2K block size. This is the recommended structure for a double sided, double track, double density disc.



The XDPBs for the standard formats are as follows:

PCW8256 FORMAT (type 0)

36	SPT, records per track
3	BSH, block shift
7	BLM, block mask
0	EXM, extent mask
174	DSM, number of blocks - 1
63	DRM, number of directory entries - 1
£C0	ALO, 2 directory blocks
£00	AL1
16	CKS, size of checksum vector
1	OFF, reserved tracks
2	PSH, physical sector shift
3	PHM, physical sector mask
0	single sided
40	tracks per side
9	sectors per track
1	first sector number
512	sector size
42	gap length (read/write)
82	gap length (format)
£60	MFM mode, skip deleted data address mark
0	do auto select format

SYSTEM FORMAT

36	SPT, records per track
3	BSH, block shift
7	BLM, block mask
0	EXM, extent mask
170	DSM, number of blocks - 1
63	DRM, number of directory entries - 1
£C0	ALO, 2 directory blocks
£00	AL1
16	CKS, size of checksum vector
2	OFF, reserved tracks
2	PSH, physical sector shift
3	PHM, physical sector mask
0	single sided
40	tracks per side
9	sectors per track
£41	first sector number
512	sector size
42	gap length (read/write)
82	gap length (format)
£60	MFM mode, skip deleted data address mark
0	do auto select format

# DATA ONLY FORMAT

36	SPT, records per track
3	BSH, block shift
7	BLM, block mask
0	EXM, extent mask
179	DSM, number of blocks - 1
63	DRM, number of directory entries - 1
£C0	AL0, 2 directory blocks
£00	AL1
16	CKS, size of checksum vector
0	OFF, reserved tracks
2	PSH, physical sector shift
3	PHM, physical sector mask
0	single sided
40	tracks per side
9	sectors per track
£C1	first sector number
512	sector size
42	gap length (read/write)
82	gap length (format)
£60	MFM mode, skip deleted data address mark
0	do auto select format

## Swapping discs between CP/M 3 and CP/M 2.2

CP/M 2.2 and AMSDOS system format and data only format discs can be used under CP/M 3 with no further ado.

CP/M 3 discs can be freely used under AMSDOS provided that:

- There is no disc label.
- There is no time and date stamping.
- There are no passwords.

This is the normal state of a CP/M 3 disc, and will remain as such unless the user explicitly enables any of the above features using INITDIR.COM and/or SET.COM.

If any of the above exist then both CP/M 2.2 and AMSDOS will become confused as to the amount of free space on the disc. Files may still be read but it is recommended that such discs are not written to under CP/M 2.2 or AMSDOS.

INITDIR.COM is used to reformat the directory for time and date stamping. SET.COM is used to create disc labels and enable and create passwords.

### Time and date clock

The CP/M 3 time and date clock in the System Control Block (SCB) is updated every second.

On the CPC6128 there are three sources of error in the clock. Firstly the +/- 0.1 tolerance on the processor clock. Secondly there are not precisely 300 ticker interrupts per second. Thirdly while accessing the disc, interrupts are disabled which, in the worst case, could lose up to 60 ticks. The first error can make the clock run fast or slow. The second error tends to make the clock run fast. The third error tends to make the clock run slow. So with the right amount of disc accesses the clock will be just right!

### Memory Organization

The CPC6128's memory is organized into three banks:

Bank 0 is the BDOS bank which contains the banked portions of the BIOS (Basic Input Output System) and BDOS (Basic Disc Operating System) it also contains the screen memory, some disc buffers and the extended BIOS jumpblock.

Bank 1 is the TPA bank in which all application programs are run.

Bank 2 contains a copy of the CCP (Console Command Processor), disc hash tables and data buffers.

The top 16K of memory is common to all banks, it contains the resident portions of the BIOS and BDOS and part of the TPA.

### Memory map

	Bank 0 - BDOS	Bank 1 - TPA	Bank 2 - Extra
	+-----+   block 7   common  -----+	+-----+   block 7   common  -----+	+-----+   block 7   common  -----+
£C000	+-----+   block 2   BIOS, BDOS   firmware jumpblock  -----+	+-----+   block 6  -----+	+-----+  -----+
£8000	+-----+   block 1   screen  -----+	+-----+   block 5  -----+	+-----+   block 3   CCP, hash tables   data buffers  -----+
£4000	+-----+   block 0 / lower ROM   more BIOS   low kernel jumpblock  -----+	+-----+   block 4  -----+	+-----+  -----+
£0000	+-----+  -----+	+-----+  -----+	+-----+  -----+

# The Implementation of CP/M Plus on the 6128 and 8256

The PCW8256's memory is organized into three banks:

Bank 0 is the BDOS bank which contains the banked portions of the BIOS (Basic Input Output System) and BDOS (Basic Disc Operating System) it also contains the screen memory, some disc buffers and the extended BIOS jumpblock.

Bank 1 is the TPA bank in which all application programs are run.

Bank 2 contains a copy of the CCP (Console Command Processor), disc hash tables, data buffers and parts of the BIOS.

The top 16K of memory is common to all banks, it contains the resident portions of the BIOS and BDOS and part of the TPA.

## Memory map

	Bank 0 - BDOS	Bank 1 - TPA	Bank 2 - Extra
	block 7   common 	block 7   common 	block 7   common 
£C000			
	block 3   BIOS, BDOS 	block 6   	
£8000			
	block 1   screen 	block 5   	block 8   CCP, hash tables   data buffers 
£4000			
	block 0   BIOS   extended jumpblock 	block 4   	
£0000			

The screen environment is not included in the above CP/M 3 bank model. The screen environment consists of blocks 7, 2, 1, 0, i.e. similar to bank 0 but with block 3 replaced by block 2 which contains the matrix RAM, roller RAM and some of the screen RAM. The screen environment can be accessed via the "SCR RUN ROUTINE" entry in the extended BIOS jumpblock.

The character matrix RAM is at £B800 in the screen environment.

The screen roller RAM is at £B600 in the screen environment.

The RAM disc is in blocks 9..15 plus any additional contiguous memory.

### The Restart Instructions and Locations

The BIOS uses the restart instructions and their locations as follows:

RST 0	: used by CP/M
RST 1..RST 5	: not used - reserved
RST 6	: used by SID
RST 7	: used for interrupts

The SID program supplied has been modified to use RST6 instead of the usual RST7.

### The Z80 Alternate and Index Registers

An application program may use the Z80 alternate and index instructions without further ado. All BDOS and standard BIOS calls preserve these registers. Extended BIOS routines preserve the alternate register set but may use the index registers for passing parameters. See the routine definitions for details.

Note that using Z80 only instructions and/or registers will prevent an application from running on an 8080 or 8085 based machine.

### Cold Boot

The term "cold boot" means invoking CP/M for the first time after the machine has been switched on or reset.

On the CPC 6128, CP/M 3 is invoked from BASIC by issuing the |CPM external command. This reads a one sector boot program into store as described in the DDI-1 firmware manual. The boot program loads the first file it finds with type '.EMS', this file should contain the CP/M 3 system.

The '.EMS' file replaces the CPM3.SYS file which is usually found on CP/M 3 implementations. The '.EMS' file contains CPM3.SYS, CCP.COM together with a loader and parts of the BIOS.

### Warm Boot

The term "warm boot" means reloading CP/M after a transient program has run or after the user has typed Control-C in order to change a disc.

A warm boot is invoked immediately after initialization and thereafter by a jump to location zero, or by calling the BDOS System Reset function. The primary function of the warm boot is to load CCP into memory and then to enter the CCP. A copy of the CCP is stored in bank 2 so a warm boot does not require any disc access.

The warm boot routine initializes the jumps at £0000 and £0005.

### BIOS User Interaction

In general the user may freely type-ahead on the keyboard, provided the BIOS is not accessing the discs. While accessing the discs interrupts are disabled and characters can be lost.

### BIOS messages

All BIOS messages are displayed on the status line, if enabled, otherwise they are sent to the CONOUT: device.

Some BIOS messages are followed by the question "Retry, Ignore or Cancel?". The system then discards any outstanding characters, turns on the cursor and waits for the user to type "R", "I" or "C", anything else will cause a bleep.

Typing "R" for retry causes the BIOS to repeat the operation.

Typing "I" for ignore causes the BIOS to continue as if the problem had not occurred.

Typing "C" for cancel causes the BIOS to abandon the operation. This will often result in a BDOS error message.

BIOS disc error messages. These are only displayed if the BDOS is not in "RETURN ERROR" mode, see BDOS function 45 "SET BDOS ERROR MODE". On two drive systems disc error messages are prefixed by A: or B: denoting the drive. On single drive systems the prefix is omitted.

"drive not ready - Retry, Ignore or Cancel? "

Usually caused by accessing a drive without a disc in it. Put a disc into the drive and press R.

"write protected - Retry, Ignore or Cancel? "

Caused by trying to write to a write-protected disc. Write enable the disc and press R, or to avoid writing to the disc press C.

"track t seek fail - Retry, Ignore, Cancel? "

Unable to seek to required track, may be caused by a drive fault. Try Retrying a few times.

"track t, sector s data error - Retry, Ignore or Cancel? "

Corrupted data, may be caused by damaged media. Try Retrying a few times if that fails then attempt to reformat the disc.

"track t, sector s no data - Retry, Ignore or Cancel? "

Cannot find required sector, may be caused by an unformatted disc. If this error persists try re-formatting the disc.

"track t, sector s missing address mark - Retry, Ignore or Cancel? "

If this error persists try re-formatting the disc.

"bad format - Retry, Ignore or Cancel? "

During login the BIOS is unable to determine the format of the disc. Disc may be unformatted or of a totally alien format.

"unknown error - Retry, Ignore or Cancel? "

Should never happen, the floppy disc controller has reported an error that is not recognized, treat like a data error.

Other messages

"CP/M Plus Amstrad Consumer Electronics plc"

"v 0.6, 61K TPA"

(v 1.0, 61K TPA on the PCW 8256)

The sign on message.

"1 disc drive"

"2 disc drives"

Indicating number of drives found. If this gives the wrong answer ensure that drive B: is empty and that the connections to it are sound.

"1 serial port"

Indicating that the serial interface is fitted.

"Drive is X:"

On single drive systems this message is always displayed on the right hand side of the status line to indicate whether the drive is currently A: or B:.

"Please put the disc for X: into the drive then press any key "

On single drive systems two "logical" drives are mapped onto the single physical drive. This message is displayed whenever a different logical drive is accessed.

"DEVICE not ready - Retry, Ignore or Cancel?"

The named output device has failed to respond for 10 seconds.

## USERF

In the standard CP/M 3 BIOS jumpblock function 30 "USERF" is reserved for the system implementor. On the CPC6128 this function is used for calling the firmware and extended BIOS routines in bank 0.

To find USERF fetch the contents of location 1, this contains the address of function 1 "WBOOT" in the BIOS jumpblock. Add 87 to give the address of the JMP USERF entry. It is a good idea to copy the USERF jumpblock entry into a fixed location and then call this fixed location.

USERF takes the address of the required routine in bank 0 as an inline parameter. The registers AF BC DE HL IX IY are all passed to the routine and returned back to the caller as set by the routine. The alternate register set is preserved throughout the call, it can neither be used to pass parameters nor to return results.

## BIOS Jumpblocks

The BIOS has two jumpblocks: the standard CP/M 3 jumpblock and an extended jumpblock. The word at location £0001 contains the address of the WBOOT entry in the standard BIOS jumpblock, function 1. The extended jumpblock starts at £0080 in bank 0 and contains jumps for additional facilities such as physical sector reading and writing.

Since the extended BIOS jumpblock is in bank 0 whereas the user program is in bank 1 it is not possible for an application program to call routines in the extended jumpblock directly, instead the BIOS function USERF must be used.

It is intended that this jumpblock will be supported on subsequent AMSTRAD CP/M 3 implementations. It has the format:

### Disc Driver

£0080	JMP DD INIT	;initialize disc driver
£0083	JMP DD SETUP	;set disc parameters
£0086	JMP DD READ SECTOR	;read a sector
£0089	JMP DD WRITE SECTOR	;write a sector
£008C	JMP DD CHECK SECTOR	;check a sector
£008F	JMP DD FORMAT	;format a track
£0092	JMP DD LOGIN	;login a disc
£0095	JMP DD SEL FORMAT	;select a standard format
£0098	JMP DD DRIVE STATUS	;fetch drive status
£009B	JMP DD READ ID	;read a sector ID
£009E	JMP DD L DPB	;initialize a DPB
£00A1	JMP DD L XDPB	;initialize an XDPB
£00A4	JMP DD L ON MOTOR	;turn motor on, wait for timeout
£00A7	JMP DD L T OFF MOTOR	;set motor off timeout
£00AA	JMP DD L OFF MOTOR	;turn motor off
£00AD	JMP DD L READ	;read type uPD765A command
£00B0	JMP DD L WRITE	;write type uPD765A command
£00B3	JMP DD L SEEK	;seek command



#### SIO Driver

```

£00B6    JMP  CD SA INIT          ;initialize SIO channel A
£00B9    JMP  CD SA BAUD         ;set SIO channel A baud rates
£00BC    JMP  CD SA PARAMS       ;fetch SIO channel A parameters

```

#### Terminal Emulator

```

£00BF    JMP  TE ASK             ;where is the cursor, what screen size
£00C2    JMP  TE RESET          ;re-initialize the screen
£00C5    JMP  TE STL ASK        ;is the status line enabled?
£00C8    JMP  TE STL ON OFF     ;enable/disable the status line
£00CB    JMP  TE SET INK        ;set the colour(s) for an ink
£00CE    JMP  TE SET BORDER     ;set the colour(s) for the border
£00D1    JMP  TE SET SPEED      ;set the ink flash speed

```

#### Keyboard

```

£00D4    JMP  KM SET EXPAND      ;set the text for an expansion token
£00D7    JMP  KM SET KEY        ;set entry(s) for key translation
£00DA    JMP  KM KT GET         ;get a key token
£00DD    JMP  KM KT PUT         ;put a key token
£00E0    JMP  KM SET SPEED      ;set key repeat speed

```

#### Misc

```

£00E3    JMP  CD VERSION        ;get version numbers
£00E6    JMP  CD INFO          ;get BIOS system information
£00E9    JMP  SCR RUN ROUTINE    ;run a routine in screen environment

```

All other entries in the jumpblock are reserved and must not be used.

The action performed by each entry in the extended jumpblock is detailed below, along with the entry and exit conditions of each.

#### 0 DD INIT (0080H)

Initializes the disc driver, resets all disc parameters to their default values. Turns the motor off. The registers AF BC DE and HL are corrupted.

The default disc parameters are:

```

Motor on timeout      1 second
Motor off timeout     5 seconds
Write current off time 1.75 milliseconds
Head settle time     30 milliseconds
Step rate            12 milliseconds
Head load time        4 millisecond
Head unload time      480 milliseconds
Non-DMA mode

```

(see also DD SETUP)

## 1 DD SETUP (0083H)

Resets various disc parameters. Sets the values for the motor on, motor off, write current off and head settle times. Sends a SPECIFY command to the floppy disc controller. On entry, HL points to the parameter block in common memory which is of the following format:-

```

bytes 0 : motor on timeout in 100 millisecond units.
bytes 1 : motor off timeout in 100 millisecond units.
byte 2 : write current off time in 10 microsecond units.
byte 3 : head settle time in 1 millisecond units.
byte 4 : step rate time in 1 millisecond units.
byte 5 : head unload delay 32..480 ms in 32 ms units
byte 6 : bits 7..1: head load delay, bit 0: non-DMA mode (as per uPD765A
        SPECIFY command).
```

On exit, AF, BC, DE and HL are corrupted, all other registers are preserved.

The values given are used for both drives. When using two differing drives use the slower of the two times. A motor off time of zero will never turn the motor off.

The default values are:

```

Motor on timeout          10
Motor off timeout         50
Write current off time    175
Head settle time          30
Step rate                 12
Head load time            £0F
Head unload time + non DMA £03
```

## 2 DD READ SECTOR (0086H)

Reads a sector from disc into any bank. On entry, the following registers must be set up:-

```

B  = CP/M bank
C  = unit
D  = logical track
E  = logical sector
HL = address of destination buffer in the bank in register B
IX = address of XDPB in common memory (£C000..£FFFF)
```

If the operation was successful, Carry is set, otherwise it is reset. The A register is corrupted. In the latter case, A has the following error code:-

```

0 means drive not ready
2 means seek fail
3 means data error
4 means no data
5 means missing address mark
8 means media changed
```

Of the other registers BC, DE and HL are corrupted.

In the event of errors this routine tries and retries a total of 15 times in such a pattern as to maximise the chances of recovery:-

T T T R T T T T I T T T T R T T T T

Where T means try, R realign, I move to innermost track.

The "media changed" error means that the sector numbers on the disc are different from those specified in the XDPB.

(see also DD WRITE SECTOR and DD CHECK SECTOR)

### 3 DD WRITE SECTOR (0089H)

Writes a sector to disc from any bank.

On entry, the registers must contain the following information:-

B = CP/M bank

C = unit

D = logical track

E = logical sector

HL = address of source buffer in bank in register B

IX = address of XDPB in common memory (&C000..&FFFF)

On exit, the carry flag is set if the operation was successful. If unsuccessful, the carry flag is set and register A is returned with one of the following:-

0 means drive not ready

1 means write protected

2 means seek fail

3 means data error

4 means no data

5 means missing address mark

8 means media changed

The "media changed" error means that the sector numbers on the disc are different from those specified in the XDPB. In the event of errors this routine tries and retries a total of 15 times using the following retry procedure:

T T T R T T T T I T T T T R T T T T

Where T means try, R realign, I move to innermost track.

BC, DE and HL are corrupted by the operation, all other registers are preserved.

(See also DD READ SECTOR and DD CHECK SECTOR)

#### 4 DD CHECK SECTOR (008CH)

Check that a sector on disc is the same as one in memory. ( Compares a sector on the disc with a store copy.)

On entry the following values need to be passed:-

B = CP/M bank

C = unit

D = logical track

E = logical sector

HL = address of source buffer in bank in register B

IX = address of XDPB in common memory (£C000..£FFFF)

If the operation was successful, then carry is set on exit, otherwise it is reset and A has one of the following values:-

0 means drive not ready

2 means seek fail

3 means data error

4 means no data

5 means missing address mark

8 means media changed

Retries are done to the pattern described above.

Zero is set if the sectors matched, otherwise it is reset.

The flags, BC, DE and HL are corrupted by the operation, all other registers preserved.

(see also DD READ SECTOR and DD WRITE SECTOR)

#### 5 DD FORMAT (008FH)

Formats a track using a header information buffer in any bank.

The registers should be set up as follows:-

B = CP/M bank

C = unit

D = logical track

E = filler byte, usually £E5

HL = address of header information buffer in bank in register B

IX = address of XDPB in common memory (£C000..£FFFF)

Format of header information:

sector entry for first sector

sector entry for second sector

...

sector entry for last sector

Sector entry format:

byte 0: track number

byte 1: head number

byte 2: sector number

byte 3: log2(sector size) - 7

If successful, carry is set. If something went wrong, then carry is reset and A has one of the following:-

- 0 means drive not ready
- 1 means write protected
- 2 means seek fail
- 3 means data error
- 4 means no data
- 5 means missing address mark
- 8 means media changed

On exit, the flags, BC, DE and HL are corrupted, all other registers are preserved. On disc error, retries are done in the manner already described.

## 6 DD LOGIN (0092H)

Login a disc. It attempts to determine the format of a disc. If successful it initializes an XDPB. It does not affect or consider the freeze flag.

On entry, the following registers should be initialised:-

C = unit

IX = address of XDPB in common memory (£C000..£FFFF)

On exit, carry is set if successful and the XDPB is initialised. A has the disc type:-

- 1 means system format
- 2 means data only format

and

DE = size of 2 bit allocation vector

HL = size of hash table

If the operation failed, then carry is reset and A contains one of the following:-

- 0 means drive not ready
- 2 means seek fail
- 3 means data error
- 4 means no data
- 5 means missing address mark
- 6 means bad format

in this case, DE HL are corrupt as well as the XDPB. Retries are conducted in the manner already described.

Whatever happens, the flags and BC are corrupt, all other registers are preserved

(see also DD SEL FORMAT)

## 7 DD SEL FORMAT (0095H)

Selects a standard disc format. It initializes an XDPB for the required format regardless of the actual disc format. Normally the BIOS automatically determines the format of a disc when it is logged in, but for programs such as disc formatters it is necessary to pre-set the format. Does not affect or consider the freeze flag.

On entry, A has one of the following values, representing a format type:-

- 1 means system format
- 2 means data only format

IX = address of XDPB in common memory (£C000..£FFFF)

On exit, BC is corrupted: carry is set if successful, and A has the disc type as follows:-

- 1 means system format
- 2 means data only format

DE has the size of the 2-bit allocation vector

HL has the size of the hash table.

and the XDPB is initialized.

If unsuccessful, carry is reset and A has a 6, meaning "bad format" (This routine will only fail if an illegal disc type is given), DE, HL and the XDPB are corrupt.

(See also DD LOGIN)

## 8 DD DRIVE STATUS (0098H)

Fetch the drive status: ready, write protected etc. On entry, C = bits 0,1: unit, bit 2: head. On exit, A = status as follows:-

- bit 7 : undefined
- bit 6 : write protected
- bit 5 : ready
- bit 4 : track 0
- bit 3 : undefined
- bit 2 : head
- bit 1,0 : unit

(For unit 1 if bit 6 = 0 and bit 5 = 0 then the drive is not fitted.)

Flags and HL are corrupted by this call.

## 9 DD READ ID (009BH)

Reads a sector ID from the first sector found. On entry:-

C = unit  
D = logical track  
IX = address of XDPB in common memory (&C000..&FFFF)

if successful, the carry flag is set and A = sector number from ID

If failed the carry flag is reset

The A reg contains the code:-

- 0 means drive not ready
- 2 means seek fail
- 3 means data error
- 4 means no data
- 5 means missing address mark

whatever the result, HL = address of results buffer in common memory

Format of results buffers

byte 0 : number of results bytes received  
bytes 1.. : results

The F, BC and DE registers are corrupted, but all other registers preserved  
Retries are done to the manner described above.

## 10 DD L DPB (009EH)

On the CPC 6128, this returns an error. On the PCW 8256 it will initialize a standard CP/M 3 Disc Parameter Block for a given disc format.

On entry,

HL = address of source disc specification in common memory (&C000..&FFFF)  
IX = address of destination DPB in common memory (&C00C..&FFFF)

On succesful exit, the carry flag is set and A = disc type; If failed the carry flag is reset and The A reg contains the code 6 meaning bad format.

Flags BC, DE and HL are corrupted, all other registers are preserved.

The disc format is specified as follows:

Byte 0: disc type  
Byte 1: sidedness  
    0 means single sided  
    1 means double sided, flip sides  
    2 means double sided, up and over  
Byte 2: number of tracks per side  
Byte 3: number of sectors per track  
Byte 4: Log2 (sector size) - 7  
Byte 5: number of reserved tracks  
Byte 6: Log2 (block size) - 7  
Byte 7: number of directory blocks  
Byte 8: gap length (read/write)  
Byte 9: gap length (format)

(See also DD L XDPB)

11 DD L XDPB (00A1H)

On the CPC 6128, this returns an error. On the PCW 8256 it will initialize an eXtended Disc Parameter Block for a given disc format.

On entry,

HL = address of source disc specification in common memory (£C000..£FFFF)

IX = address of destination XDPB in common memory (£C000..£FFFF)

on exit,

If successful, then the carry flag is set

A = disc type

If failed, then the carry flag is reset

The A reg contains the code 6 (means bad format)

BC, DE and HL are corrupted, but all other registers preserved

The disc format is specified as follows:

Byte 0: disc type

Byte 1: sidedness

0 means single sided

1 means double sided, flip sides

2 means double sided, up and over

Byte 2: number of tracks per side

Byte 3: number of sectors per track

Byte 4: Log2 (sector size) - 7

Byte 5: number of reserved tracks

Byte 6: Log2 (block size) - 7

Byte 7: number of directory blocks

Byte 8: gap length (read/write)

Byte 9: gap length (format)

(see DD L DPB)

12 DD L ON MOTOR (00A4H)

If the motor is off then turn it on, wait for the motor on timeout.

On exit, all registers and flags are preserved

(See also DD L T OFF MOTOR and DD L OFF MOTOR)



13 DD L T OFF MOTOR (00A7H)

Starts the motor off timeout, after which the motor will be turned off. Does not wait for the timeout.

All registers and flags preserved on exit.

(see also DD L ON MOTOR and DD L OFF MOTOR)

14 DD L OFF MOTOR (00AAH)

Turns the motor off, kills the motor ticker if any.

Corrupts AF, BC, DE and HL. All other registers are preserved

(see also DD L ON MOTOR and DD L T OFF MOTOR)

15 DD L READ (00ADH)

uPD765A read type command driver. This is the Low level interface for the uPD765A floppy disc driver. Use for "READ DATA", "READ DELETED DATA", "READ A TRACK". It sends the required commands to the uPD765A, deals with bank switching, fetches results. The motor must be running. Detailed knowledge of the uPD765A is required in order to use this routine. On the CPC6128, sector commands are terminated by setting EOT to the required sector number, this produces End of Cylinder errors which should be ignored.

On entry, HL = address of parameter block in common memory.

Format of parameter block:

- byte 0 : CP/M bank
- byte 1,2 : address of buffer
- byte 3,4 : number of bytes to transfer
- byte 5 : number of uPD765A command bytes
- byte 6.. : command bytes

on exit, HL = address of results buffer in common memory.

Format of results buffers

- byte 0 : number of results bytes received
- bytes 1.. : results

The registers AF, BC and DE are corrupted. All other registers preserved.

(See also DD L WRITE)

# 16 DD L WRITE (00B0H)

uPD765A write type command driver. This is the low level interface for the uPD765A floppy disc driver. Use for "WRITE DATA", "WRITE DELETED DATA", "FORMAT A TRACK", "SCAN EQUAL", "SCAN LOW OR EQUAL", "SCAN HIGH OR EQUAL". It sends required commands to the uPD765A, deals with bank switching, fetches results. The Motor must be running. Detailed knowledge of the uPD765A is required in order to use this routine. On the CPC6128 sector commands are terminated by setting EOT to the required sector number, this produces End of Cylinder errors which should be ignored.

On entry, HL = address of parameter block in common memory

Format of parameter block:

```
byte 0   : CP/M bank
byte 1,2 : address of buffer
byte 3,4 : number of bytes to transfer
byte 5   : number of uPD765A command bytes
byte 6.. : command bytes
```

on exit, HL = address of results buffer in common memory.

Format of results buffers

```
byte 0   : number of results bytes received
bytes 1.. : results
```

the registers AF, BC and DE are corrupted, all other registers are preserved.

(See also DD L READ)

# 17 DD L SEEK (00B3H)

Seeks to the required track, realigns if required, seeks to given track. the motor must be running.

On entry:-

```
C = bits 0,1: unit, bit 2: head
D = physical track
IX = address of XDPB in common memory
```

```
on successful exit, the carry flag is set and A is corrupt
If failed, the carry flag is reset and the A reg contains the code:-
    0 means not ready
    2 means seek fail
```

The routine always returns zero true and the other flags, B and IY are corrupt. All other registers are preserved. The routine will try 10 times to seek or realign before returning an error.

This routine is not normally required since "DD READ SECTOR", "DD WRITE SECTOR" "DD CHECK SECTOR" and "DD FORMAT" all perform their own seeks.

## 18 CD SA INIT (00B6H)

Initialize SIO channel A.

On entry,

A = mode

£00 means no handshake

£FF means handshake

D = number of stop bits

0 means 1

1 means 1.5

2 means 2

E = parity

0 means none

1 means odd

2 means even

H = number of receive data bits 5, 6, 7 or 8

L = number of transmitter data bits 5, 6, 7 or 8

on exit; AF, BC, DE and HL are corrupted, but all other registers preserved

(The parameters are not validated, silly values will give silly results.)

## 19 CD SA BAUD (00B9H)

Set the baud rates for channel A of the SIO. Sets the receiver and transmitter baud rates for SIO channel A.

On entry:-

H = encoded receiver baud rate

L = encoded transmitter baud rate

on exit,

AF, BC, DE and HL are corrupted, but all other registers are preserved

(The parameters are not validated, silly values will give silly results.

This call does not affect the CP/M 3 character I/O table.)

## 20 CD SA PARAMS (00BCH)

Fetches the current mode, parity, baud rates etc for SIO channel A

on exit:-

A = mode

£00 means non-handshake

£FF means handshake

B = receiver encoded baud rate

C = transmitter encoded baud rate

D = stop bits

0 means 1

1 means 1.5

2 means 2

E = parity

0 means none

1 means odd  
2 means even

H = receiver data bits 5, 6, 7 or 8  
L = transmitter data bits 5, 6, 7 or 8

All other registers preserved

## 21 TE ASK (00BFH)

Fetches the current cursor position and screen size.(and viewpoint position on the PCW 8256). On the CPC 6128, The screen size given is the bottom right hand corner of the terminal emulators screen, not the physical screen size. The size will depend on whether or not 24 x 80 mode is enabled or the status line is enabled.

The top row is row 0, the left column is column 0.

On exit:-

B = top row of the viewpoint in physical screen coordinates (8256 only)  
C = left column of the viewpoint in physical screen coordinates (8256 only)  
D = bottom row of screen ( height of viewport - 1 on the PCW 8256)  
E = right column of screen ( width of viewport - 1 on the PCW 8256)  
H = cursor row (in viewport coordinates on the PCW 8256)  
L = cursor column (in viewport coordinates on the PCW 8256)

All other registers preserved

## 22 TE RESET (00C2H)

Re-initializes the terminal emulator: sets mode 2, clears the screen, homes and enables the cursor, resets the inks to their standard colours. For use by programs which have written to the screen by means other than using the CRT device.

Registers AF, BC, DE and HL are corrupted, all other registers are preserved.

## 23 TE STL ASK (00C5)

Asks if the status line is enabled.

If enabled, returns with zero false; If disabled, zero true.

The carry flag is always reset, A is corrupted, all other registers preserved

(see also TE STL ON OFF)

24 TE STL ON OFF (00C8H)

Enables or disables the status line. Disabling the status line gives an extra line to the terminal emulator. When disabled status line messages are sent to the CONOUT: device.

On entry, A = enable/disable.

£00 means disable

£FF means enable

on exit: F, BC, DE and HL are corrupted, all other registers are preserved.

(see also TE STL ASK)

25 TE SET INK (00CBH)

Set which two colours will be used to display an ink. If the two colours are the same then the ink will remain a steady colour. If the two colours are different then the ink will alternate between these colours. On the PCW8256 there are only two inks 0 and 1, and two colours black and "white". If colour of ink 0 is greater than the colour of ink 1 then the whole screen is displayed with black characters on "white" background, otherwise "white" characters on a black background.

The ink number is masked with 0FH to make sure it is legal (01H on the PCW 8256). The colours are masked with 3FH and the resulting value is treated as three 2 bit numbers each specifying the intensity of one of the three primary colours. Bits 0,1 for blue, bits 2,3 for red and bits 4,5 for green. On the CPC6128 the three levels of intensity are mapped as follows:

Colour parameter :	0	1	2	3	
CPC6128	:	0	1	1	2

If ink 0 is specified then the border is also changed to the same colours.

On entry:-

A contains an ink number

B contains the first colour

C contains the second colour

on exit: AF, BC, DE and HL are corrupted; all other registers are preserved.

26 TE SET BORDER (00CEH)

Set which two colours will be used to display the border. If the two colours are the same then the border will remain a steady colour. If the two colours are different then the border will alternate between these colours.

The colours are masked with £3F and the resulting value is treated as three 2 bit numbers each specifying the intensity of one of the three primary colours. Bits 0,1 for blue, bits 2,3 for red and bits 4,5 for green. On the CPC6128 the three levels of intensity are mapped as follows:

Colour parameter :	0	1	2	3
CPC6128 :	0	1	1	2

The border can also be changed by TE SET INK.

On entry:-

B contains the first colour  
C contains the second colour

On exit, AF, BC, DE and HL are corrupted, but all other registers are preserved.

27 TE SET SPEED (00D1H)

Set the flash period, for how long each of the two colours for the inks and the border are to be displayed on the screen. These settings apply to all inks and the border.

The flash periods are given in frame flybacks (1/50 or 1/60 of a second). A period of 0 is taken to mean a period of 256.

The default setting for the flash periods is 10 frame flybacks (1/5 or 1/6 of a second).

The new flash periods are not used immediately but when the inks next flash.

On entry,

H contains the period for the first colour  
L contains the period for the second colour

on exit,

AF and HL are corrupted, all other registers are preserved.

28 KM SET EXPAND (00D4H)

Set the expansion string associated with an expansion token. The characters in the string are not expanded (or otherwise dealt with). It is therefore possible to put any character into an expansion string.

If there is insufficient room in the expansion buffer for the new string then no change is made to the expansions.

If the string is currently being used to generate characters then the unread portion of the string is discarded. The next character will be read from the key buffer.

On entry,

B contains the expansion token for the expansion to set

C contains the length of the string

HL contains the address of the string in common memory (£C000..£FFFF)

On exit, the carry flag is set if the expansion is OK.

If the string was too long or the token was invalid the carry flag is reset

Whatever the outcome, other flags, A, BC, DE and HL are corrupted, all other registers preserved

29 KM SET KEY (00D7H)

Set entry(s) in key translation table(s), what character or token a key will be translated to when shift, or control, or neither is pressed.

If the key number is invalid (greater than 79) then no action is taken.

Most values in the table are treated as characters and are passed back to the user. However, there are certain special values:

£80..£9F are the expansion tokens and are expanded to character strings.

£FD is the caps lock token and causes the caps lock to turn on if it is off, or vice versa.

£FE is the shift lock token and changes the shift state on/off.

£FF is the ignore token and means the key should be thrown away.

On entry,

B contains the new translation

C contains a key number

D contains bit mask indicating which table, or tables, to be changed

bit 0 means normal translation, neither shift nor control pressed

bit 1 means shift translation

bit 2 means control translation

other bits ignored

On exit, AF and HL are corrupted, all other registers are preserved.

30 KM KT GET (00DAH)

Try to fetch a key token from the keyboard. The shift state defines which locks and/or shift keys were pressed. The repeat bit indicates that the key was generated by a repeat.

If got a token then the carry flag is set

C = character

B = 0 (CPC 6128 only).

If no token available then the carry flag is reset

BC corrupt

Other flags and A are always corrupted.

B = shift state (only on PCW 8256)

bit 0 means not defined

bit 1 means extra

bit 2 means caps lock

bit 3 means repeat

bit 4 means num lock

bit 5 means shift

bit 6 means shift lock

bit 7 means alt

All other registers preserved

31 KM KT PUT (00DD)

Put a key token. (PCW 8256 only). Inserts a key token into the keyboard buffer so that the next "KM KT GET" will fetch it. More than one key token may be put, however, buffer overflow is not reported and will result in key tokens being lost. The buffer is at least 10 tokens long. If a great deal of text is to be generated by KM KT PUT the use of expansion tokens is recommended.

To change either the caps lock or num lock states call "KM KT PUT" with the required state with an innocuous key number then call "KM KT GET" to remove it. The shift lock state cannot be changed in this way as it is hardware controlled.

On entry:-

B = shift state

bit 0 means not defined

bit 1 means extra

bit 2 means caps lock

bit 3 means repeat

bit 4 means num lock

bit 5 means shift

bit 6 means shift lock

bit 7 means alt

C = key number

All registers and flags are preserved on exit.



32 KM SET SPEED (00E0H)

Set the time before keys first repeat (start up delay) and the time between repeats (repeat speed). Both delays are given in scans of the keyboard. The keyboard is scanned every fiftieth of a second.

A start up delay or repeat speed of 0 is taken to mean 256.

The default start up delay is 30 scans (0.6 seconds) and the default repeat speed is 2 scans (0.04 seconds or 25 characters a second).

Note that a key is prevented from repeating (by the key scanner) if the key buffer is not empty. Thus the actual repeat speed is the slower of the supplied repeat speed and the rate at which characters are removed from the buffer. This is intended to prevent the user from getting too far ahead of a program that is running sluggishly.

The start up delay and repeat speed apply to all keys on the keyboard that are set to repeat.

On entry,  
H contains the new start up delay.  
L contains the new repeat speed.

On exit, AF is corrupted, all other registers are preserved.

33 CD VERSION (00E3H)

Fetches machine type, BIOS version numbers and machine specific version numbers.

On exit:-

A = machine  
    0 means CPC6128  
    1 means PCW8256  
B = BIOS major version number  
C = BIOS minor version number  
DE reserved  
HL = machine specific version number  
    CPC6128  
    H = on board ROM's version number  
    L = on board ROM's mark number  
    PCW8256  
    not defined

All other registers are preserved

### 34 CD INFO (00E6H)

Fetches BIOS system information, number of disc drives, address of buffer table, number of memory blocks etc. The buffer table gives details of where the data and directory buffer areas are.

On exit,

A = number of disc drives  
 £00 means 1 disc drive  
 £FF means 2 disc drives  
 B = number of memory blocks  
 C = serial interface status  
 £00 means not fitted  
 £FF means fitted  
 HL = address of buffer table in common memory (£C000..£FFFF)

Format of buffer table

entry for buffer area 0  
 entry for buffer area 1  
 ...  
 £FF

Entry format:

byte 0 : CP/M bank  
 byte 1,2 : start address  
 byte 3,4 : size in bytes

DE corrupt.

All other registers preserved

### 35 SCR RUN ROUTINE (00E9H)

(PCW 8256 only) Runs a routine in the screen environment. Switches memory blocks 7, 2, 1, 0 into context, these blocks contain the character matrix RAM, the roller RAM and the screen RAM. Then calls the supplied routine. On exit the memory context is restored to its original state.

The screen is 720 pixels wide and 256 pixels high. Let (0,0) be the top left corner; (0,719) the top right corner; (255,0) bottom left corner; and (255,719) the bottom right corner.

The character matrix RAM is at B800H and has the following format:

byte 0: character 0  
 byte 8: character 1  
 ...  
 byte 2040: character 255

Each character entry has the following format:

byte 0: pixel row 0  
byte 1: pixel row 1  
...  
byte 7: pixel row 7

Each pixel row has the following format:

bit 0: pixel column 7  
bit 1: pixel column 6  
...  
bit 7: pixel column 0

The roller RAM is at £B600 and has the following format:

bytes 0, 1: address of pixel row 0  
bytes 2, 3: address of pixel row 1  
...  
bytes 510, 511: address of pixel row 255

Each pixel row has the following format:

byte 0: pixel columns 0..7  
byte 8: pixel columns 8..15  
...  
byte 712: pixel columns 712..719

Each pixel column byte has the following format:

bit 0: pixel column 7  
bit 1: pixel column 6  
...  
bit 7: pixel column 0

On entry:-

BC = address of routine to call, in common memory (£C000..£FFFF)  
AF DE HL IX IY as required by routine

on exit, AF, DE, HL, IX and IY are as returned by the called routine

BC is corrupted but all other registers are preserved.

CALLing the Firmware on the CPC6128 from CP/M 3

A firmware routine can only be called by using USERF. Even so, only a subset of firmware routines may be called. They are listed below. Using USERF, or accessing the hardware directly, stops an application program from being portable.

Some of the firmware routines require the address of a parameter block. This parameter block must be in common memory, i.e. £C000 and above. If a firmware routine returns an address this will be an address in bank 0. It is, therefore, likely to be only useful to an application program as a parameter to a further firmware routine.

The upper ROM must not be enabled. The screen is at £4000 in bank 0, it must not be moved.

The terminal emulator (CRT device) uses the TXT and SCR routines. If an application program also wishes to use these or the GRA routines certain steps must be taken: Before using any of the TXT, GRA or SCR routines the terminal emulator's cursor must be disabled by sending ESC f. This is because the cursor is turned on by a ticker which could go off whilst the application program was already in the firmware giving unpredictable results.

If the status line is enabled all status line messages will be displayed on the status line regardless of the current CONOUT: device. If the status line is disabled status line messages are sent to the CONOUT: device. Thus to avoid status line messages appearing on the screen disable the status line and redirect the CONOUT: device away from the CRT device. Alternatively avoid any action which could cause a status line message such as disc errors or selecting logical drive B: on a single drive system.

Device redirection can be performed using the DEVICE.COM utility or by writing to the indirection vectors in the System Control Block (SCB) using BDOS function 49, see the chapter on the BDOS.

If it is required to use the CRT device as well as the TXT, SCR or GRA routines then any mode changing should be done by sending Esc 3 mode to the CRT device, this informs the terminal emulator of the size of the screen etc. Before calling any BIOS or BDOS routine which could cause characters to be sent to the CRT device, ensure that the cursor position is where the CRT device last had it, that the window covers the whole screen, and that inverse video is as the CRT device last had it.

The status line may still be used if required.

When finished restore the screen to its original state using TE RESET (or by other means) and, if necessary, redirect the CONOUT: device to the CRT.

# Summary of firmware calls and restrictions on the CPC 6128

In this summary "OK" means that the entry can be called using USERF without any more ado. However, an application should take care to restore the machine back to the state in which it found it, otherwise the BIOS screen and keyboard drivers may become confused.

Where addresses are required to be in common memory this is indicated by "Entry HL >= £C000".

Where an address is returned in bank 0 this is indicated by "address in bank 0".

"BANNED" means the entry cannot be used, this restriction is not enforced - break it at your peril!

## Main Firmware Jumpblock

0	KM INITIALISE	OK	
1	KM RESET	OK	
2	KM WAIT CHAR	OK	
3	KM READ CHAR	OK	
4	KM CHAR RETURN	OK	
5	KM SET EXPAND	Entry HL >= £C000	(but see extended jumpblock)
6	KM GET EXPAND	OK	
7	KM EXP BUFFER	Entry DE >= £C000	
8	KM WAIT KEY	OK	
9	KM READ KEY	OK	(but see extended jumpblock)
10	KM TEST KEY	OK	
11	KM GET STATE	OK	
12	KM GET JOYSTICK	OK	
13	KM SET TRANSLATE	OK	(but see extended jumpblock)
14	KM GET TRANSLATE	OK	
15	KM SET SHIFT	OK	(but see extended jumpblock)
16	KM GET SHIFT	OK	
17	KM SET CONTROL	OK	(but see extended jumpblock)
18	KM GET CONTROL	OK	
19	KM SET REPEAT	OK	
20	KM GET REPEAT	OK	
21	KM SET DELAY	OK	(but see extended jumpblock)
22	KM GET DELAY	OK	
23	KM ARM BREAKS	BANNED	
24	KM DISARM BREAK	OK	
25	KM BREAK EVENT	OK	
26	TXT INITIALIZE	OK	
27	TXT RESET	OK	
28	TXT VDU ENABLE	OK	
29	TXT VDU DISABLE	OK	
30	TXT OUTPUT	OK	
31	TXT WR CHAR	OK	
32	TXT RD CHAR	OK	
33	TXT SET GRAPHIC	OK	
34	TXT WIN ENABLE	OK	
35	TXT GET WINDOW	OK	
36	TXT CLEAR WINDOW	OK	
37	TXT SET COLUMN	OK	
38	TXT SET ROW	OK	

# The Implementation of CP/M Plus on the 6128 and 8256

39	TXT SET CURSOR	OK
40	TXT GET CURSOR	OK
41	TXT CUR ENABLE	OK
42	TXT CUR DISABLE	OK
43	TXT CUR ON	OK
44	TXT CUR OFF	OK
45	TXT VALIDATE	OK
46	TXT PLACE CURSOR	OK
47	TXT REMOVE CURSOR	OK
48	TXT SET PEN	OK
49	TXT GET PEN	OK
50	TXT SET PAPER	OK
51	TXT GET PAPER	OK
52	TXT INVERSE	OK
53	TXT SET BACK	OK
54	TXT GET BACK	OK
55	TXT GET MATRIX	Exit HL address in bank 0
56	TXT SET MATRIX	Entry HL >= £C000
57	TXT SET M TABLE	Entry HL >= £C000, Exit HL address in bank 0
58	TXT GET M TABLE	Exit HL address in bank 0
59	TXT GET CONTROLS	Exit HL address in bank 0
60	TXT STR SELECT	OK
61	TXT SWAP STREAMS	OK
62	GRA INITIALISE	OK
63	GRA RESET	OK
64	GRA MOVE ABSOLUTE	OK
65	GRA MOVE RELATIVE	OK
66	GRA ASK CURSOR	OK
67	GRA SET ORIGIN	OK
68	GRA GET ORIGIN	OK
69	GRA WIN WIDTH	OK
70	GRA WIN HEIGHT	OK
71	GRA GET W WIDTH	OK
72	GRA GET W HEIGHT	OK
73	GRA CLEAR WINDOW	OK
74	GRA SET PEN	OK
75	GRA GET PEN	OK
76	GRA SET PAPER	OK
77	GRA GET PAPER	OK
78	GRA PLOT ABSOLUTE	OK
79	GRA PLOT RELATIVE	OK
80	GRA TEST ABSOLUTE	OK
81	GRA TEST RELATIVE	OK
82	GRA LINE ABSOLUTE	OK
83	GRA LINE RELATIVE	OK
84	GRA WR CHAR	OK
85	SCR INITIALISE	BANNED
86	SCR RESET	OK
87	SCR SET OFFSET	OK
88	SCR SET BASE	BANNED
89	SCR SET LOCATION	OK
90	SRC SET MODE	OK (or send Esc 3 mode to the CRT)
91	SCR GET MODE	OK
92	SCR CLEAR	OK
93	SCR CHAR LIMITS	OK
94	SCR CHAR POSITION	Exit HL address in bank 0

95	SCR DOT POSITION	Exit HL address in bank 0
96	SCR NEXT BYTE	Exit HL address in bank 0
97	SCR PREV BYTE	Entry and Exit HL address in bank 0
98	SCR NEXT LINE	Entry and Exit HL address in bank 0
99	SCR PREV LINE	Entry and Exit HL address in bank 0
100	SCR INK ENCODE	OK
101	SCR INK DECODE	OK
102	SCR SET INK	OK (but see extended jumpblock)
103	SCR GET INK	OK
104	SCR SET BORDER	OK (but see extended jumpblock)
105	SCR GET BORDER	OK
106	SCR SET FLASHING	OK (but see extended jumpblock)
107	SCR GET FLASHING	OK
108	SCR FILL BOX	OK
109	SCR FLOOD BOX	Entry HL address in bank 0
110	SCR CHAR INVERT	OK
111	SCR HW ROLL	OK
112	SCR SW ROLL	OK
113	SCR UNPACK	Entry HL address in bank 0, DE >= £C000
114	SCR REPACK	Entry DE address in bank 0
115	SCR ACCESS	OK
116	SCR PIXELS	Entry HL address in bank 0
117	SCR HORIZONTAL	OK
118	SCR VERTICAL	OK
119	CAS INITIALISE	OK
120	CAS SET SPEED	OK
121	CAS NOISY	OK
122	CAS START MOTOR	OK
123	CAS STOP MOTOR	OK
124	CAS RESTORE MOTOR	OK
125	CAS IN OPEN	Entry DE, HL >= £C000, Exit HL address in bank 0
126	CAS IN CLOSE	OK
127	CAS IN ABANDON	OK
128	CAS IN CHAR	OK
129	CAS IN DIRECT	Entry HL >= £C000
130	CAS RETURN	OK
131	CAS TEST EOF	OK
132	CAS OUT OPEN	Entry DE, HL >= £C000, Exit HL address in bank 0
133	CAS OUT CLOSE	OK
134	CAS OUT ABANDON	OK
135	CAS OUT CHAR	OK
136	CAS OUT DIRECT	Entry HL >= £C000
137	CAS CATALOG	Entry DE >= £C000
138	CAS WRITE	Entry HL >= £C000
139	CAS READ	Entry HL >= £C000
140	CAS CHECK	Entry HL >= £C000
141	SOUND RESET	OK
142	SOUND QUEUE	Entry HL >= £C000
143	SOUND CHECK	OK
144	SOUND ARM EVENT	BANNED
145	SOUND RELEASE	OK
146	SOUND HOLD	OK
147	SOUND CONTINUE	OK
148	SOUND AMPL ENVELOPE	Entry HL >= £C000
149	SOUND TONE ENVELOPE	Entry HL >= £C000
150	SOUND A ADDRESS	Exit HL address in bank 0

The Implementation of CP/M Plus on the 6128 and 8256

151	SOUND T ADDRESS	Exit HL address in bank 0
152	KL CHOKE OFF	BANNED
153	KL ROM WALK	BANNED
154	KL INIT BACK	BANNED
155	KL LOG EXT	BANNED
156	KL FIND COMMAND	BANNED
157	KL NEW FRAME FLY	BANNED
158	KL ADD FRAME FLY	BANNED
159	KL DEL FRAME FLY	BANNED
160	KL NEW FAST TICKER	BANNED
161	KL ADD FAST TICKER	BANNED
162	KL DEL FAST TICKER	BANNED
163	KL ADD TICKER	BANNED
164	KL DEL TICKER	BANNED
165	KL INIT EVENT	BANNED
166	KL EVENT	BANNED
167	KL SYNC RESET	BANNED
168	KL DEL SYNCHRONOUS	BANNED
169	KL NEXT SYNC	BANNED
170	KL DO SYNC	BANNED
171	KL DONE SYNC	BANNED
172	KL EVENT DISABLE	BANNED
173	KL EVENT ENABLE	BANNED
174	KL DISARM EVENT	BANNED
175	KL TIME PLEASE	OK
176	KL TIME SET	OK
177	MC BOOT PROGRAM	BANNED
178	MC START PROGRAM	BANNED
179	MC WAIT FLYBACK	OK
180	MC SET MODE	BANNED
181	MC SCREEN OFFSET	BANNED
182	MC CLEAR INKS	Entry DE >= £C000
183	MC SET INKS	Entry DE >= £C000
184	MC RESET PRINTER	OK
185	MC PRINT CHAR	OK
186	MC BUSY PRINTER	OK
187	MC SEND PRINTER	OK
188	MC SOUND REGISTER	OK
189	JUMP RESTORE	BANNED
190	KM SET LOCKS	OK
191	KM FLUSH	OK
192	TXT ASK STATE	OK
193	GRA DEFAULT	OK
194	GRA SET BACK	OK
195	GRA SET FIRST	OK
196	GRA SET LINE MASK	OK
197	GRA FROM USER	OK
198	GRA FILL	Entry HL >= £C000
199	SCR SET POSITION	BANNED
200	MC PRINT TRANSLATION	Entry HL >= £C000
201	MC BANK SELECT	BANNED



Indirection Jumpblock

BANNED

High Kernel Jumpblock

BANNED

Low Kernel Jumpblock

BANNED

## Appendix D – The Amstrad Utilities

There are a number of utilities supplied with the operating system that are not part of CP/M itself. They are supplied with the Amstrad PCW8256/ CPC 6128 to give access to the hardware-specific facilities.

LANGUAGE.COM	selects a character set
PALETTE.COM	sets the ink colours
PAPER.COM	initializes an Epson FX-80 or PCW8256 printer
SETSIO.COM	sets the parameters of the SIO (if fitted)
SETKEYS.COM	configures the keyboard
SETLST.COM	sends data to initialize the printer
SET24X80.COM	sets the screen into 24 x 80 mode

These utilities are primarily designed to be used from the PROFILE.SUB startup file although they also may be used at any time as required. They are neither interactive nor easy to use, and are not recommended for use by the beginner.

In the parameter tail of these utilities, numbers may be expressed in decimal or in hexadecimal. Hexadecimal numbers are preceded by a £ or &. All other numbers are assumed to be in decimal. Characters may be either upper or lower case. Words and numbers should be separated by spaces, commas or other innocuous characters such as =.

# LANGUAGE.COM

This utility selects one of eight international character sets and effects the way that characters appear on the screen.

it is used by typing

LANGUAGE n

where n is a number from 0 to 7 to give the required language number as follows:

- |   |         |
|---|---------|
| 0 | U.S.A.  |
| 1 | France  |
| 2 | Germany |
| 3 | U.K.    |
| 4 | Denmark |
| 5 | Sweden  |
| 6 | Italy   |
| 7 | Spain   |

#### PALETTE.COM

This utility sets any or all of the inks to the required colours.  
it is used by typing:-  
PALETTE number number number ...

The first number specifies the colour of ink 0, the second number specifies ink 1 and so on until either all inks have been specified or the list of colours is exhausted.

Each colour is given as a number in the range 0..63. The colour number represents three 2 bit numbers each corresponding to the intensity of one of the primary colours, bits 4,5 for green, bits 2,3 for red and bits 0,1 for blue.

On the CPC6128, there are 15 inks and three levels of colour intensity; these are mapped onto the required four levels of intensity as follows:

```
colour : 0 1 2 3
CPC6128 : 0 1 1 2
```

intensity 3 is interpreted as intensity 2.

The PCW8256 has a monochromatic screen. There are two inks 0 and 1. If the colour of ink 0 is greater than ink 1 then the screen is displayed in inverse video, black characters on a white background, otherwise white characters on a black background.

Any colour number greater than 63 is masked with 63.

If more colours than inks are given the remainder are ignored.

#### Examples

```
palette 0           sets ink 0 to black
palette £30 £0C     sets ink 0 bright green, ink 1 bright red
```

## PAPER.COM

Set printer parameters for the PCW8256 printer or an Epson FX-80.

It is used by typing:-

PAPER parameter parameter ...

To use the printer effectively it is necessary to tell it the length of paper in use, whether it is single sheet or continuous, and so on. The PAPER utility allows the operator to set these parameters.

PAPER is a program which takes a number of parameters from the command line which invokes it. Once the parameters have been checked for validity the program sends suitable escape sequences to the printer (via the CP/M LST: calls). Note that this means that firstly it may be used with almost any Epson compatible printer and secondly it requires the printer to be ready to accept characters. After each escape sequence is sent to the printer PAPER reports on the console what it has sent.

The parameters for PAPER are :

Form Length <number>

The <number> must be in the range 6..99, and sets the form length in lines. If the Line Pitch is not set explicitly in this use of PAPER, then it is set to "standard line pitch" (six lines per inch).

If a Gap Length is not set explicitly in this use of PAPER then the gap length is set to zero.

Gap Length <number>

The <number> must be in the range 0..99, and sets the gap length in lines. If the gap length specified is not zero and the Line Pitch is not set explicitly in this use of PAPER, then it is set to "standard line pitch" (six lines per inch).

Line Pitch <number>

The <number> may be 6 or 8, setting 6 or 8 lines per inch.

Single Sheet

Sets single sheet stationery. If Paper Out Defeat is not set explicitly in this use of PAPER then it is set On.

Continuous Stationery

Sets continous stationery. If Paper Out Defeat is not set explicitly in this use of PAPER then it is set Off.

Paper Out Defeat On or Paper Out Defeat Off

Sets paper out defeat as required.

Defaults

Tells the printer to copy its current settings (including those

## Appendix D - The Amstrad Utilities

given in this use of PAPER) to its memory of default settings.

The program in fact requires only the first letter of each of the keywords given above, except for On and Off which must be given in full. In all cases only the first keyword is required, the others are optional.

Three further parameters are accepted :

A4 or A5

These set : 6 lines per inch  
Form length 70 lines (A4) or 50 lines (A5)  
Gap length 3 lines  
Single Sheet  
Paper out defeat On

<number>

This must be a number in the range 1..17 and is provided to set up for continuous stationery, with a form length as given measured in inches.

The following are set : 6 lines per inch  
Form length <number> inches  
Gap length 0  
Continuous stationery  
Paper out defeat Off

## SETSIO.COM

This utility initializes the SIO serial interface.

### Syntax

SETSIO option option option ...

Where an option is any, or all, of the following in any order:

#### Baud rate

TX 300	sets transmitter baud rate to 300 baud
RX 134.5	sets receiver baud rate to 134.5 baud
9600	sets both baud rates to 9600 baud

The baud rate must be one of 50, 75, 110, 134.5, 150, 300, 600, 1200, 1800, 2400, 3600, 4800, 7200, 9600, 19200.

#### Number of data bits

BITS 7                      7 data bits

The number of data bits must be one of 5, 6, 7 or 8.

#### Number of stop bits

STOP 1                      1 stop bit

The number of stop bits must be one of 1, 1.5 or 2.

#### Parity

PARITY EVEN	sets even parity
PARITY ODD	sets odd parity
PARITY NONE	sets no parity

#### XON protocol

XON ON	enables XON protocol
XON OFF	disables XON protocol

#### Control signal handshake

HANDSHAKE ON	enabled handshake
HANDSHAKE OFF	disables handshake

For the words TX, RX, STOP, BITS, PARITY, XON and HANDSHAKE only the initial letter is required, the remainder of the word is ignored.

### Semantics

The SIO is initialized as required. For baud rates and the XON options the CP/M 3 character I/O table entry will be changed. The SIO is device number.

The baud rate option without a preceding RX or TX sets both baud rates.

If the baud rate is specified but the number of stop bits is not specified

then the number of stop bits is set to 1 if the baud rate is greater than 110 otherwise it is set to 2. Any other option not specified retains its original value.

If an option is given more than once, or if both baud rate and TX baud rate etc are given then the later option in the list is used. For example "SETSIO 9600, 300" will set the baud rate to 300.

Any illegal option will produce an error message and the option will be ignored.

After the SIO has been initialized the current state of the SIO settings is displayed in the same form as the command as follows:

9600 Bits 8 Parity none Stop 1 Xon off Handshake on

Thus the command SETSIO may be used to examine the current settings.

#### Examples

SETSIO PARITY EVEN

SETSIO

SETSIO 9600, P NONE, HANDSHAKE=ON, STOP 2, BITS 5



## SETKEYS.COM

This utility reconfigures the keyboard as required. A command file is required which contains the keyboard configuration data.

### Syntax

SETKEYS filename

The command file contains the keyboard configuration data and has the following syntax.

Each line contains a key definition or an expansion token definition. A key definition associates a key in a given shift state, or states, with a character or token value. An expansion token definition associated an expansion token with a string.

#### Key definition

A key definition consists of a key number optionally followed by a shift state or states, followed by the required character in quotes. Any other characters on the same line are treated as comment.

```
71 "z" lower case Z
71 S "Z" upper case Z
```

The shift states are machine dependent:

CPC6128

S for shift.  
C for control.  
N or nothing to indicate no shift.

PWC8256

S for shift.  
A for alt.  
E for extra.  
SA for shift and alt.  
N or nothing to indicate no shift.

More than one shift state may be given in which case the definition will apply to all the shift states.

The character associated with the key is either the character itself or an escape sequence.

Characters in the range £20..£FF other than ^ or " stand for themselves. ^ introduces an escape sequence. Control codes must be represented by escape sequences.

^ followed by a character in the range £40..£FF masks the character with £1F thus ^A gives Control A.

^^ gives the character ^.

^" gives the character ".

^ followed by a number enclosed by single quotes gives a character of that value. ^'£D' gives carriage return.

^ followed by the name of a control code enclosed by single quotes gives that control code, ^'ESC' gives the ESC code.

The names of the control codes are NUL, SOH, STX, ETX, EOT, ENQ, ACK, BEL, BS, HT, LF, VT, FF, CR, SO, SI, DLE, DC1, DC2, DC3, DC4, NAK, SYN, ETB, CAN, EM, SUB, ESC, FS, GS, RS, US, SP, DEL, XON, XOFF.

#### Examples

```
67 "q"           lower case Q
67 S "Q"         upper case Q
67 C "^Q"        control Q

66 N S C "^'ESC'"  escape is always escape
```

#### Expansion Token Definitions

An expansion token definition consists of E followed by the token number followed by the expansion string enclosed by "", followed by a comment if required. The characters in the string are represented by the same characters and escape sequences as in key definitions.

Token numbers are machine dependent:

```
CPC6128: £80..£9F.
PCW8256: £80..£9E. (£9F is the ignore token).
```

Any key can be set to one of these values in which case it will return the expansion string.

#### Examples

```
E £80 "DIR B: [SIZE]^M"
E £87 "MALLARD^M"
```

In an expansion token definition bit 7 of the token value is ignored.

When parsing the command file any line which contains an error is displayed on the console with an error message and ignored, parsing continues with the next line.

## SETLST.COM

This utility sends a string of characters to the LST: device in order to initialize the printer. A command file is required which contains the information to send to the printer. Note that this file is not sent directly to the printer but is interpreted as described below.

## Syntax

SETLST filename

The command file contains the information to send to the LST: device. All the information is represented by characters in the range £20..£FF. Control codes are ignored. To send control codes to the printer escape sequences must be used as follows.

Characters in the range £20..£FF other than ^ stand for themselves. ^ introduces an escape sequence.

- ^ followed by a character in the range £40..£FF masks the character with £1F thus ^A gives control-A.
- ^^ gives the character ^.
- ^" gives the character ".
- ^ followed by a number enclosed by single quotes gives a character of that value. ^'£D' gives carriage return.
- ^ followed by the name of a control code enclosed by single quotes gives that control code, ^'ESC' gives the ESC code.

The names of the control characters are NUL, SOH, STX, ETX, EOT, ENQ, ACK, BEL, BS, HT, LF, VT, FF, CR, SO, SI, DLE, DC1, DC2, DC3, DC4, NAK, SYN, ETB, CAN, EM, SUB, ESC, FS, GS, RS, US, SP, DEL, XON, XOFF.

Any illegal escape sequence will produce an error message and the sequence will be ignored.

## Examples:

```
^'ESC'A5563
^Abcd
^'STX'^'£F4'
```

**SET24X80.COM**

Some application programs may require a standard 24 x 80 screen. This utility sets the screen size to 24 x 80 regardless of the actual screen size.

Syntax

SET24X80 ON  
SET24X80 OFF

Semantics

SET24X80 ON sets 24 x 80 mode. SET24X80 OFF restores the screen to its full size. The full size depends on the machine, the country and whether or not the status line is enabled.

If the parameter is omitted then ON is assumed.

## Appendix E

### CCP.COM disassembled

CCP.COM the console command processor disassembled to illustrate the method used by CP/M Plus to load programs, provide multiple line commands, load and delete RSXs, and other system functions which may be of interest to advanced programmers. With this information, a programmer could replace the CCP with an application dependent menu, thereby removing any direct interface between the operator and the CP/M Plus operating system.

With this information, the author has produced a screen command processor called SHELL to replace the CCP interface. SHELL fills the screen with a directory, and surrounds this by cursor key selection of executable commands. SHELL is much less inhibiting to the first time user.

```

                                title      'Dis-assembled CCP.COM using DISZ80.COM'
                                .z80

000D      cr      equ      0dh
000A      lf      equ      0ah

0000'                                aseg
                                org      0000h

;-----
; Page Zero areas      ;      SET - Initialised by CCP
;-----
0000      WBOOT  EQU      $      ;
0004      SET_DSK EQU      $+0004H ;      Set to default DSK/USR
0005      BDOS   EQU      $+0005H ;
0050      SET_DR  EQU      $+0050H ;      Set to FCB.DR of file loaded
0051      SET_PA1 EQU      $+0051H ;      Set to password of first filename
0053      SET_PL1 EQU      $+0053H ;      Set to length of first password
0054      SET_PA2 EQU      $+0054H ;      Set to password of 2nd filename
0056      SET_PL2 EQU      $+0056H ;      Set to length of 2nd password
005C      SET_FCB EQU      $+005CH ;      Set to first filename
006C      SET_F2  EQU      $+006CH ;      Set to second filename
0080      SET_CMD EQU      $+0080H ;      Set to command line tail
;-----

                                org      0100h

;-----
; TPA load address
;-----
0100      tpa    EQU      $

;-----
; CP/M Plus BDOS SYSTEM CONTROL BLOCK
;-----
0064      lenscb equ      100      ;
009C      scb    equ      low -lenscb ; Under 3.1 LOW(SCB) is always 9CH
;-----
0090      xscb   equ      scb-12    ; Xtended SCB
0090      xsch00 equ      scb-12    ; CCP sets to zero before overlay load
0091      xsch01 equ      scb-11    ; CCP sets to zero before overlay load
0092      xsch02 equ      scb-10    ; CCP sets to zero before overlay load
0093      xsch03 equ      scb-9     ; CCP sets to zero before overlay load
0098      xsch08 equ      scb-4     ; Address of BDOS (word)
;-----
00A1      scb_05 equ      scb+(0A1h-09ch) ; BDOS version number
00AB      scb_0F equ      scb+(0ABh-09ch) ; Reserved
00AC      scb_10 equ      scb+(0ACb-09ch) ; Program Error return code (2 bytes)
00AE      scb_12 equ      scb+(0AEb-09ch) ; Reserved - Page of multiple line RSX
00AF      scb_13 equ      scb+(0AFb-09ch) ; Reserved - Default Disk
00B0      scb_14 equ      scb+(0B0b-09ch) ; Reserved - Default User Number
00B1      scb_15 equ      scb+(0B1b-09ch) ; Reserved - Ptr to multiple line RSX
00B3      scb_17 equ      scb+(0B3b-09ch) ; Reserved
00B4      scb_18 equ      scb+(0B4b-09ch) ; Reserved
00B5      scb_19 equ      scb+(0B5b-09ch) ; Reserved
00B6      scb_1A equ      scb+(0B6b-09ch) ; Console Width
00B8      scb_1C equ      scb+(0B8b-09ch) ; Console Page Length
00BA      scb_1E equ      scb+(0BAh-09ch) ; Reserved - Address of text

```

```

00BC      scb_20 equ    scb+(0BCh-09ch) ; Reserved - Address of text
00C8      scb_2C equ    scb+(0C8h-09ch) ; Page Mode
00C9      scb_2D equ    scb+(0C9h-09ch) ; Reserved
00CF      scb_33 equ    scb+(0CFh-09ch) ; Console Mode (2 bytes)
00D3      scb_37 equ    scb+(0D3h-09ch) ; Output Delimiter
00DA      scb_3E equ    scb+(0DAh-09ch) ; Current Disk
00E0      scb_44 equ    scb+(0E0h-09ch) ; Current User Number
00E5      scb_4A equ    scb+(0E5h-09ch) ; BDOS Multi-Sector Count
00E7      scb_4B equ    scb+(0E7h-09ch) ; BDOS Error Mode
00E8      scb_4C equ    scb+(0E8h-09ch) ; Drive Search Chain (4 bytes)
00EC      scb_50 equ    scb+(0ECh-09ch) ; Temporary File Drive
00F9      scb_5D equ    scb+(0F9h-09ch) ; Common Memory Base Address (2 bytes)
00F9      scb_62 equ    scb+(0F9h-09ch) ; Reserved (2 bytes) OS Base Address

```

```

; scb_17 bit flags
; Bit 0 - Set by BDOS if submit file $$$SUB is on directory
;         - Cleared when file $$$SUB is deleted
; Bit 1 - Loader Sets if COM program starts with a RET
;         (GENCOM defines NULL COM program)
; Bit 2 - Set as part of CCP SCB initialisation
; Bit 6 - Set if COM program starts with a RET
;         - Cleared before calling LOADER FUNCTION 59 - LOAD OVERLAY
;         - if BIT 7 set
;         - If set USER set to current not default
;         - and DISK set to current disk not OFFH
; Bit 7 - Set by BDOS FUNCTION 47, and use command line at 0080h
;         - Cleared before calling LOADER FUNCTION 59 - LOAD OVERLAY

```

```

; scb_18 bits
; Bit 3-4 = 00 - No TYPE assumed
;           = 01 - ASSUME COM if none specified
;           = 10 - ASSUME SUB if none specified
;           = 11 - ASSUME PRL if none specified
; Bit 5 - If set CCP resets disk
;         - Set before CCP prompt, Cleared after CCP line entered
; Bit 6 - If set CCP does not copy SCB 2D to SCB 2C (Page Mode)
; Bit 7 - Set before CCP prompt, Cleared after CCP line entered
;         - except cleared before CCP line read if SCB 1E/1F contains
;         - valid text address from SCB 15/16 (multiple line RSX)

```

```

; scb_19 bits
; Bit 0 - Set if submit file active
; Bit 1 - Set after first time CCP loaded (first time PROFILE.S run)

```

```

; scb_10 (word)
; - Cleared to 0000H before calling BDOS FUNCTION 59 if
;   SCB_17 BIT 7 zero

```

```

; CCP program entry and
; Start of LOADER RSX

```

```

0100
0000'   C3 031A'
0003'

```

```

cseg
CCP:    jp      START
        ds      3

```

```

0006' C3 001B'    LOADER: jp    LOAD_MAIN      ; RSX START
0009' C3 0006     nx_rsx: jp    BDOS+1         ; RSX NEXT
000C' 0007       pr_rsx: dw    BDOS+2         ; RSX PREV
000E' 00 00      DB    0,0                   ; RSX REMOVE & NONBANK FLAGS
0010' 4C 4F 41 44    DB    'LOADER '        ; RSX NAME
0014' 45 52 20 20    DB
0018' FF          DB    0FFH                 ; RSX LOADER FLAG
0019' 00 00      DB    0,0                   ; RSX RESERVED AREA
001B'             LOAD_MAIN:
001B' 79          ld     a,c
001C' FE 3B      cp     59                     ; BDOS Load Overlay
001E' C2 0009'    jp     nz,nx_rsx

;-----;
; (BDOS) FUNCTION 59 - LOAD OVERLAY (COM or PRL with or without RSX) ;
;-----;
; On entry:
; C = 3EH
; DE = 'opened' FCB with bytes R0 R1 -> load address, or
;     = 0000h
; (SP) = 0100H if loaded by CCP.COM
; If DE = 0000h, then no file read, but RSX'S deleted
; If (SP) = 0100h and loaded file does not contain RSX'S, then
;     jump vector at 6,7 and at SCB offset 99 moved above loader
; Returns:
; If (SP) = 0100H, then executes loaded program if successful
;     or displays error message and warm boots
; if (SP) ^= 0100H, then
;     A = 00H if successful
;     A = 0FFH if bad address or no memory
;     A = 0FFH if physical error and extended errors enabled
; NB If the program loaded overlays the calling program, then the
; address in the stack (SP) should be set to a valid (i.e. 0100H)
; address.
;-----;

0021' C1          pop     bc                    ; <BC> = return address
0022' C5          push    bc
0023' 21 0000     ld     hl,0
0026' 39          add     hl,sp
0027' 31 02BE'    ld     sp,load_sp
002A' 22 029A'    ld     (save_sp),hl
002D' C5          push    bc                    ; Place CALLERS return address on STACK
002E' EB          ex      de,hl
002F' 22 0298'    ld     (save_de),hl
0032' 7C          ld     a,h
0033' B5          or      l
0034' F5          push    af
0035' CC 0100'    call    z,era_rsx             ; if DE = 0 then erase RSX's
0038' F1          pop     af
0039' C4 0130'    call    nz,load_ovly         ; DE > 0
003C' D1          pop     de                    ; recover CALLERS address from stack
003D' 21 0100     ld     hl,tpa
0040' 7E          ld     a,(hl)
0041' FE C9      cp     0C9H                    ; Test for RSX at 0C9H
0043' CA 009E'    jp     z,load_GENCOM

```



```

0046' 7A          ld    a,d          ; test for return address of 0100H
0047' 3D          dec    a
0048' B3          or     e
0049' C2 005F'    jp     nz,ret_ok    ; -no-

004C' 3A 000D'    ld     a,(pr_rsx + 1) ; test for RSX loaded
004E' B7          or     a          ; by testing HIGH (6,7)
0050' C2 005F'    jp     nz,ret_ok    ; -yes-

```

```

;-----
; 'REMOVE' loader from TPA as
; 1. Return address = 0100H
; 2. and no RSX loaded
;-----

```

```

0053' 2A 000A'    ld     hl,(rx_rsx+1)
0056' 22 0006     ld     (BDCG+1),hl
0059' 22 0294'    ld     (OS_BASE),hl
005C' CD 00F8'    call    set_62

```

```

;-----
; load operation successful
; Return A = 0 & H = 0
;-----

```

```

005F' 2A 029A'    ret_ok: ld     hl,(save_sp)
0062' F9          ld     sp,hl
0063' AF          xor     a
0064' 6F          ld     l,a
0065' 67          ld     h,a
0066' C9          ret

```

; This may return to loaded program

```

;-----
; load operation failed
; Return A = FEH & H = 0
; or as in DE
;-----

```

```

0067' mem_err:    11 00FE    ld     de,00FEH    ; bad load address or no memory
006A' err_load:   2A 029A'    ld     hl,(save_sp)
006D' F9          ld     sp,hl
006E' E1          pop     hl
006F' E5          push    hl
0070' 25          dec     h
0071' 7C          ld     a,h
0072' B5          or     l
0073' EB          ex      de,hl
0074' 7D          ld     a,l
0075' 44          ld     b,h
0076' C0          ret     nz          ; return if return address not 0100H

```

```

0077' Bad_load:  0E 09      ld     c,9          ; Print String
0079' 11 0253'    ld     de,msg_bl
007C' CD 0005     call    BDCG
007F' C3 0000     jp     WBOOT

```

;-----

```

; LOAD RSX
; File containing an RSX is preceeded
; with a header record created by GENCOM
;-----
load_RSX:
0082'      23          inc     hl          ; GENCOM_16+2
0083'      4E          ld      c,(hl)
0084'      23          inc     hl          ; GENCOM_16+3
0085'      46          ld      b,(hl)      ; BC = length of MODULE
0086'      3A 028F'    ld      a,(sav_scb_5E) ; test HIGH COMMON BASE
0089'      B7          or      a
008A'      CA 0092'    jp      z,l_rsx1    ; -unbanked system-
008D'      23          inc     hl          ; GENCOM_16+4
008E'      34          inc     (hl)
008F'      CA 009D'    jp      z,l_rsx2
l_rsx1:    push     de          ; save GENCOM_16+0 word
0092'      D5          call    top_less_b  ; return DE = load address for RSX
0093'      CD 020F'    call    top_less_b
0096'      E1          pop      hl
0097'      CD 01CA'    call    page_reloc  ; from HL to DE length BC
009A'      CD 00D0'    call    init_rsx
009D'      E1          l_rsx2: pop     hl    ; recover next 16 byte header

load_GENCOM:
009E'      11 0010    ld      de,16      ; offset in GENCOM header
00A1'      19          add     hl,de
00A2'      E5          push    hl
00A3'      5E          ld      e,(hl)      ; GENCOM_16+0 word
00A4'      23          inc     hl
00A5'      56          ld      d,(hl)
00A6'      7B          ld      a,e
00A7'      B2          or      d
00A8'      C2 0082'    jp      nz,load_RSX
00AB'      CD 0103    call    tpa + 0003H  ; what does this do ? - Sets SCB etc.
00AE'      3A 0200    ld      a,(tpa+0100h)
00B1'      FE C3      cp      0C9H        ; test for COM program having a RET
00B3'      C2 00BF'    jp      nz,l_gen1
00B6'      2A 028D'    ld      hl,(PTR_SCB)
00B9'      2E B3      ld      l,scb_17
00BB'      7E          ld      a,(hl)
00BC'      F6 02      or      0000010B    ; yes so set scb_17 bit1
00BE'      77          ld      (hl),a
00BF'      2A 0101    l_gen1: ld      hl,(TPA+1) ; length to load
00C2'      44          ld      b,h
00C3'      4D          ld      c,l
00C4'      21 0200    ld      hl,tpa+0100h
00C7'      11 0100    ld      de,tpa
00CA'      CD 0226'    call    ldir
00CD'      C3 005F'    jp      ret_ok

;-----
; Initialise RSX header items
;-----
init_rsx:
00D0'      2A 0006    ld      hl,(BDCS+1)  ; address of next RSX or this loader
00D3'      2E 00      ld      l,0
00D5'      01 0006    ld      bc,6        ; copy serial number into RSX

```

```

00D8' 0D 0226'      call  ldir
00DB' 1E 18          ld     e,018H      ; set RSX offset LOADER FLAG
00DD' 12             ld     (de),a      ; = 0
00DE' 1E 0D          ld     e,00DH      ; set RSX offset PREV HIGH
00ED' 12             ld     (de),a      ; = 0
00E1' 1B             dec     de         ; set RSX offset PREV LOW
00E2' 3E 07          ld     a,7         ; (i.e. into page zero)
00E4' 12             ld     (de),a      ; = 7
00E5' 6B             ld     l,e
00E6' 1E 0B          ld     e,00BH
00E8' 73             ld     (hl),e      ; set NEXT RSX PREV LOW = 00BH
00E9' 23             inc     hl
00EA' 72             ld     (hl),d      ; set NEXT RSX PREV HIGH = HIGH RSX
00EB'               link_rsx:
00EB' EB             ex      de,hl
00EC' 72             ld     (hl),d      ; set RSX OFFSET NEXT HIGH = HIGH NEXT RSX
00ED' 2B             dec     hl
00EE' 36 06          ld     (hl),6      ; set RSX OFFSET NEXT LOW = 06H
00F0'               enable_rsx:
00F0' 2E 06          ld     l,6
00F2' 22 0006        ld     (BDOS+1),hl ; set page zero to point to RSX
00F5' 22 0294'        ld     (OS_BASE),hl ; and dont forget this as well
00F8'               set_62:
00F8' 11 0292'        ld     de,scb_62_pb ; set SCB (6,7) to HL
00FB' 0E 31          ld_scb: ld     c,49 ; Get/Set SCB
00FD' C3 0005        jp      BDOS

;-----
; Remove any removable RSX's
;-----

0100'               era_rsx:
0100' 2A 0006        ld     hl,(BDOS+1)
0103' 44             ld     b,h
0104'               next_rsx:
0104' 60             ld     h,b
0105' 2E 18          ld     l,018H      ; test byte at offset 0018H in BDOS
0107' 34             inc     (hl)       ; this is RSX loader flag
0108' 35             dec     (hl)
0109' C0             ret     nz         ; and is 0FFH for loader, 0 for any other RSX

; Have RSX
010A' 2E 0B          ld     l,00BH      ; HIGH next RSX module
010C' 46             ld     b,(hl)      ; fetch HIGH byte at offset 000BH
010D' 2E 0E          ld     l,00EH      ; REMOVE FLAG
010F' 7E             ld     a,(hl)      ; if 0FFH then removed from memory
0110' B7             or     a
0111' CA 0104'        jp     z,next_rsx ; dont remove

; Remove RSX from MEMORY
0114' 2E 0C          ld     l,00CH      ; LOW previous module
0116' 5E             ld     e,(hl)
0117' 23             inc     hl
0118' 56             ld     d,(hl)      ; <DE> = address of previous RSX module or 00007H
0119' 78             ld     a,b        ; B = HIGH address of next RSX
011A' 12             ld     (de),a
011B' 1B             dec     de
011C' 3E 06          ld     a,6        ; LOW address is always 06H
011E' 12             ld     (de),a

```

```

011F' 13          inc    de
0120' 60          ld     h,b
0121' 2E 0C       ld     1,00CH      ; set PREV of next RSX to PREV address
0123' 73          ld     (hl),e
0124' 23          inc    hl
0125' 72          ld     (hl),d
0126' 7A          ld     a,d
0127' B7          or     a           ; test for PREV -> ZERO PAGE
0128' C5          push   bc
0129' CC 00F0'    call   z,enable_rsx ; YES
012C' C1          pop    bc
012D' C3 0104'    jp     next_rsx

```

```

;-----
; Load overlay from FCB already opened
;-----

```

```

0130'          load_ovly:
0130' E5          push   hl
0131' 11 0290'    ld     de,sch_3c_pb
0134' CD 00FB'    call   ld_scb      ; Get Current DMA address
0137' EB          ex     de,hl
0138' E1          pop    hl
0139' E5          push   hl          ; save FCB
013A' 01 0020     ld     bc,32
013D' 09          add    hl,bc
013E' 36 00       ld     (hl),0      ; set FCB_CR = 0
0140' 23          inc    hl
0141' 4E          ld     c,(hl)      ; r0
0142' 23          inc    hl
0143' 66          ld     h,(hl)      ; r1
0144' 69          ld     l,c          ; hl = load address
0145' 25          dec    h
0146' 24          inc    h
0147' CA 0067'    jp     z,mem_err   ; r1 = 01h
014A' E5          push   hl          ; save load address
014B' D5          push   de          ; save current DMA address
014C' E5          push   hl
014D' CD 0231'    call   set_scb_4A  ; BDOS Multi-Sector Count
0150' E1          pop    hl
0151' F5          push   af          ; save previous Multi-Sector Count
0152' 1E 80       ld     e,128       ; maximum number of sectors
0154'          load_more:
0154' 3A 0007     ld     a,(BDOS+2)
0157' 3D          dec    a           ; A = page below BDOS
0158' 94          sub    h
0159' DA 01FA'    jp     c,load4     ; Load PAGE is > BDOS
015C' 3C          inc    a
015D' FE 40       cp     040H
015F' D2 0176'    jp     nc,load2    ; Load PAGE is > 16K of BDOS
0162' 07          rlca              ; Load PAGE is within 4000H of BDOS
0163' 5F          ld     e,a
0164' 7D          ld     a,l
0165' B7          or     a
0166' CA 0176'    jp     z,load2     ; if L = 0 Set Multi Sector count = pages * 2
0169' 06 02       ld     b,2
016B' 3D          dec    a

```

```

016C' FA 0170'      jp      m,load1      ; L is 2 sector lengths
016F' 05           dec      b            ; L is 1 sector length
0170' 7B          load1: ld      a,e
0171' 90           sub      b
0172' CA 01FA'      jp      z,load4      ; Load address overlaps BDOS
0175' 5F          ld      e,a          ; Set Multi Sector count = pages * 2 - B
0176' D5          load2: push    de
0177' E5          push    hl
0178' CD 0233'      call    put_scb_4A    ; BDOS Multi-Sector Count
017B' E1          pop     hl
017C' E5          push    hl
017D' CD 023B'      call    ms_read
0180' E1          pop     hl
0181' D1          pop     de          ; Recover Multi Sector count used
0182' F5          push    af          ; save error code
0183' 7B          ld      a,e
0184' 3C          inc     a
0185' 1F          rra
0186' 84          add     a,h
0187' 67          ld      h,a          ; adjust H by sectors read
0188' 22 0296'      ld      (top_ovly),hl ; (ASSUMES WHOLE PAGES)
018B' F1          pop     af
018C' CA 0154'      jp      z,load_more   ; no errors
018F' C1          load3: pop     bc      ; recover previous Multi-Sector Count
0190' 3D          dec     a            ; error code at EOF
0191' 58          ld      e,b
0192' CD 0233'      call    put_scb_4A    ; BDOS Multi-Sector Count
0195' 0E 1A        ld      c,26         ; Set DMA address
0197' D1          pop     de          ; recover previous current DMA address
0198' F5          push    af
0199' CD 0005      call    BDOS
019C' F1          pop     af
019D' 2A 029C'      ld      hl,(phys_err)
01A0' EB          ex      de,hl
01A1' C2 006A'      jp      nz,err_load   ; not at EOF
01A4' D1          pop     de          ; recover load address
01A5' E1          pop     hl          ; recover FCB
01A6' 01 0009      ld      bc,9         ; offset to file type
01A9' 09          add     hl,bc
01AA' 7E          ld      a,(hl)
01AB' E6 7F        and     07FH        ; mask attribute bit
01AD' FE 50        cp      'P'
01AF' C0          ret     nz
01B0' 23          inc     hl
01B1' 7E          ld      a,(hl)
01B2' E6 7F        and     07FH
01B4' FE 52        cp      'R'
01B6' C0          ret     nz
01B7' 23          inc     hl
01B8' 7E          ld      a,(hl)
01B9' E6 7F        and     07FH
01BB' D6 4C        sub     'L'
01BD' C0          ret     nz
;-----
; loaded FRL
;-----

```

```

01BE' 7B          ld    a,e          ; load address
01BF' B7          or     a           ; must be on page boundary
01C0' C2 0067'    jp     nz,mem_err
01C3' 62          ld     h,d
01C4' 6B          ld     l,e
01C5' 23          inc    hl
01C6' 4E          ld     c,(hl)
01C7' 23          inc    hl
01C8' 46          ld     b,(hl)      ; BC = program size
01C9' 6B          ld     l,e
;-----
; Page relocation
; HL -> PRL type header
; DE -> load address
; BC -> length of program
;-----
01CA'             page_reloc:
01CA' 24          inc     h           ; to start of page
01CB' D5          push    de          ; load address
01CC' C5          push    bc          ; program size
01CD' CD 0226'    call    ldir
01D0' C1          pop     bc
01D1' D1          pop     de
01D2' D5          push    de          ; load address
01D3' 5A          ld     e,d
01D4' 1D          dec     e
01D5' E5          push    hl          ; start of bit map
01D6' 63          ld     h,e          ; H = load address - 01h (page relocation)
01D7' 1E 00       ld     e,0          ; DE = load address
01D9'             prl_bit:
01D9' 78          ld     a,b
01DA' B1          or     c
01DB' CA 01F7'    jp     z,done_prl   ; length of program
01DE' 0B          dec     bc          ; -1
01DF' 7B          ld     a,e
01E0' E5 07       and     7
01E2' C2 01EA'    jp     nz,prl_bit1
01E5' E3          ex      (sp),hl
01E6' 7E          ld     a,(hl)      ; hl -> bit map
01E7' 23          inc    hl
01E8' E3          ex      (sp),hl
01E9' 6F          ld     l,a
01EA'             prl_bit1:
01EA' 7D          ld     a,l
01EB' 17          rla
01EC' 6F          ld     l,a
01ED' D2 01F3'    jp     nc,prl_bit2  ; 0 = no page relocation
01F0' 1A          ld     a,(de)
01F1' 84          add     a,h          ; relocate by H
01F2' 12          ld     (de),a
01F3'             prl_bit2:
01F3' 13          inc     de
01F4' C3 01D9'    jp     prl_bit
01F7'             done_prl:
01F7' D1          pop     de
01F8' D1          pop     de

```

```

01F9' C9                ret

01FA' CD 0231'          load4: call set_scb_4A      ; BDOS Multi-Sector Count
01FD' 21 0080            ld hl,SET_CMD
0200' CD 023B'          call ms_read
0203' C2 018F'          jp nz,load3
0206' 21 00FE            ld hl,00FEH
0209' 22 029C'          ld (phys_err),hl
020C' C3 018F'          jp load3

;
; Return DE -> address of page BC bytes below page address at 0007h
; also checks that page not less than 0F00H and not less than top_ovly
;
020F' top_less_b:
020F' 3A 0007            ld a,(BDOS+2)
0212' 3D                dec a
0213' 0B                dec bc
0214' 90                sub b
0215' 03                inc bc
0216' FE 0F            cp 00FH
0218' DA 0077'          jp c,Bad_load
021B' 2A 0296'          ld hl,(top_ovly)
021E' BC                cp h
021F' DA 0077'          jp c,Bad_load
0222' 57                ld d,a
0223' 1E 00            ld e,0
0225' C9                ret

0226' 78                ldir: ld a,b
0227' B1                or c
0228' C8                ret z
0229' 0B                dec bc
022A' 7E                ld a,(hl)
022B' 12                ld (de),a
022C' 13                inc de
022D' 23                inc hl
022E' C3 0226'          jp ldir

0231' set_scb_4A:
0231' 1E 01            ld e,1      ; set Multi-Sector Count to 1
0233' put_scb_4A:
0233' 2A 028D'          ld hl,(PTR_SCB)
0236' 2E B5            ld l,scb_4A
0238' 7E                ld a,(hl)      ; BDOS Multi-Sector Count
0239' 73                ld (hl),e      ; set to <E>
023A' C9                ret          ; returns previous value in <A>

023B' ms_read:
023B' EB                ex de,hl
023C' 0E 1A            ld c,26      ; Set DMA address
023E' E5                push hl
023F' CD 0005            call BDOS
0242' 0E 14            ld c,20      ; Read Sequential
0244' 2A 0298'          ld hl,(save_de)
0247' EB                ex de,hl

```

```

0248' 0D 0005          call  BDOS
024B' 22 029C'         ld      (phys_err),hl
024E' D1               pop     de          ; recover E = multi sector count
024F' B7               or      a
0250' C8               ret      z
0251' 5C               ld      e,h          ; set E = sectors read
0252' C9               ret

0253' 0D               msg_bl: DB      cr          ; p.s. this message is not correctly terminated
0254' 0A               DB      lf
0255' 43 61 6E 6E      DB      'Cannot load Program 221282  CCPM8 ' '82 DRI '
0259' 6F 74 20 6C
025D' 6F 61 64 20
0261' 50 72 6F 67
0265' 72 61 6D 20
0269' 32 32 31 32
026D' 38 32 20 20
0271' 43 4F 50 59
0275' 52 20 27 38
0279' 32 20 44 52
027D' 49 20
027F'                  DS      14
028D' 0000             PTR_SCB: DW      00000H
028F' 00               sav_scb_5E: DB      000H          ; HIGH common page (0000 = unbanked)

0290'                  scb_3c_pb:
0290' 3C               DB      03CH
0291' 00               DB      000H          ; GET current DMA address

0292'                  scb_62_pb:
0292' 62               DB      062H          ; SET OS_BASE in reserved area
0293' FE               DB      0FEH          ; (word)
                                ;continues overlaying bit map

                                ;-----;
                                ; PRL BIT map for loader RSX ;
                                ;-----;

0294'                  BITMAP equ      $

0294' 00 80 00 00      DB      00H,80H,00H,00H,80H,48H,41H,10H
0298' 80 48 41 10
029C' 04 12 24 12      DB      04H,12H,24H,12H,40H,08H,00H,10H
02A0' 40 08 00 10
02A4' 00 88 44 48      DB      00H,88H,44H,48H,00H,20H,04H,80H
02A8' 00 20 04 80
02AC' 00 09 00 20      DB      00H,09H,00H,20H,00H,00H,01H,20H
02B0' 00 00 01 20
02B4' 00 00 10 00      DB      00H,00H,10H,00H,00H,11H,12H,00H
02B8' 00 11 12 00
02BC' 00 41 00 10      DB      00H,41H,00H,10H,40H,82H,08H,21H
02C0' 40 82 08 21
02C4' 00 22 08 01      DB      00H,22H,08H,01H,10H,00H,00H,00H
02C8' 10 00 00 00
02CC' 20 01 00 04      DB      20H,01H,00H,04H,08H,01H,02H,08H
02D0' 08 01 02 08
02D4' 24 12 00 24      DB      24H,12H,00H,24H,40H,00H,84H,00H

```



```

02D8' 40 00 84 00
02DC' 02 04 00 00          DB  02H,04H,00H,00H,00H,00H,00H,00H
02E0' 00 00 00 00
02E4' 00 00 00 1A          DB  00H,00H,00H,1AH,1AH,1AH,1AH,1AH
02E8' 1A 1A 1A 1A
02EC' 1A 1A 1A 1A          DB  1AH,1AH,1AH,1AH,1AH,1AH,1AH,1AH
02F0' 1A 1A 1A 1A
02F4' 1A 1A 1A 1A          DB  1AH,1AH,1AH,1AH,1AH,1AH,1AH,1AH
02F8' 1A 1A 1A 1A
02FC' 1A 1A 1A 1A          DB  1AH,1AH,1AH,1AH

```

```

;-----
; Data Area overlays BITMAP
;-----

```

```

0294' OS_BASE      equ  BITMAP          ; lowest of RSX_E or LOADER_E or BDOS_E
0296' top_ovly     equ  OS_BASE + 2      ; Location above top of overlay loaded
0298' save_de      equ  top_ovly + 2      ; save callers DE here
029A' save_sp      equ  save_de + 2       ; save callers SP here
029C' phys_err     equ  save_sp + 2       ; save BDOS physical error here (WORD)
029E' load_stack   equ  phys_err + 2
02BE' load_sp      equ  load_stack + 32   ; 16 levels of stack

```

```

;----- end of loader program (page boundary) -----

```

```

0300'          ds      10

030A' CA 0A58'      patch: jp  z,c_not4      ; NULL character
030D' FE 20          cp  ' '                ; SPACE
030F' CA 0A58'      jp  z,c_not4
0312' FE 09          cp  9                  ; TAB
0314' CA 0A58'      jp  z,c_not4
0317' C3 0A53'      jp  c_nxt4

```

```

;-----
; Start of CCP process
;-----

```

```

031A' 31 0E2D'      START: ld  sp,top_sp
031D' 21 040C'      ld  hl,ccp_prompt
0320' E5            push hl

0321' 11 0C6A'      ld  de,SCB_3A          ; SCB_PB to get word at offset 3A
0324' 0E 31        ld  c,49                ; Get/Set SCB
0326' CD 0005       call BDOS
0329' 22 028D'      ld  (PTR_SCB),hl       ; save base address of SCB

032C' 2E FA        ld  l,scb_5D+1          ; HIGH common page
032E' 7E          ld  a,(hl)
032F' 32 028F'      ld  (sav_scb_5E),a
0332' 2E 99        ld  l,xscb08+1          ; Data 3 bytes BELOW SCB - HIGH BDOS_E
0334' 7E          ld  a,(hl)
0335' 32 0CA0'      ld  (bdos_pg),a

0338' 3A 0007       ld  a,(BDOS+2)          ; Does zero page
033B' 96          sub  (hl)                ; point to BDOS_E

```

```

033C'  C2 0358'          jp      nz,rdy59          ; -no- so RSX's and/or loader RSX are present
;=====
; Relocate loader below BDOS
;=====
033F'  01 0294          ld      bc,BITMAP-CCP
0342'  CD 020F'          call    top_less_b        ; NB this tests TOP_OVLY - currently bit map
0345'  63              ld      h,e                ; (returns E = 0, & D = page)
0346'  6B              ld      l,e
0347'  CD 01CA'          call    page_reloc        ; from HL to DE length BC
034A'  2A 0006          ld      hl,(BDOS+1)
034D'  6B              ld      l,e
034E'  0E 06           ld      c,6
0350'  CD 0AAE'          call    copy_c            ; copies serial number
0353'  1E 0B           ld      e,00FH
0355'  CD 00EB'          call    link_rsx

0358'
0358'  0E 62           rdY59: ld      c,98          ; Free temporary blocks
035A'  CD 0005          call    BDOS

;=====
; Read and initialise SCB
;=====
035D'  06 B6           ld      b,scb_1A
035F'  CD 0AFF'          call    get_scb          ; Return A = Console Width (base 0)
0362'  3C              inc      a                ; +1
0363'  0F              rrca                     ; /2
0364'  0F              rrca                     ; /4
0365'  0F              rrca                     ; /8
0366'  0F              rrca                     ; /16
0367'  EB 0F           and      00FH
0369'  11 0C97'        ld      de,mod_scb_1A
036C'  12              ld      (de),a            ; number of 16 byte fields in console width
036D'  2E B8           ld      l,scb_1C
036F'  7E              ld      a,(hl)            ; Console Page Length
0370'  3D              dec      a
0371'  13              inc      de
0372'  12              ld      (de),a
0373'  AF              xor      a
0374'  13              inc      de
0375'  12              ld      (de),a            ; set y_curs to zero

0376'  3E 24           ld      a,'S'            ; Default terminator
0378'  13              inc      de
0379'  12              ld      (de),a

037A'  2E D3           ld      l,scb_37
037C'  77              ld      (hl),a
037D'  2E E5           ld      l,scb_4A
037F'  36 01           ld      (hl),1          ; Default Multi Sector Count
0381'  23              inc      hl
0382'  AF              xor      a
0383'  77              ld      (hl),a            ; scb_4B Disk error mode
0384'  2E CF           ld      l,scb_33
0386'  36 01           ld      (hl),1          ; Default Console Mode
0388'  23              inc      hl

```

```

0389' 77          ld      (hl),a          ; 2nd byte
038A' 2E A1       ld      1,scb_05
038C' 36 31       ld      (hl),031H      ; CP/M version number
038E' 2E B4       ld      1,scb_18
0390' 7E         ld      a,(hl)
0391' E5 20       and     00100000B
0393' 0E 0D       ld      c,13          ; Reset Disk System
0395' E5         push    hl
0396' C4 0005     call    nz,BDOS        ; if scb_18 bit 5 set
0399' E1         pop     hl
039A' 2E B3       ld      1,scb_17
039C' 7E         ld      a,(hl)
039D' E5 02       and     00000010B      ; test for NULL COM file loaded
039F' E5         push    hl
03A0' CC 0100'    call    z,era_rsx      ; -NO-
03A3' E1         pop     hl
03A4' 7E         ld      a,(hl)          ; scb_17
03A5' E5 FD       and     11111101B      ; clear NULL COM file flag
03A7' 77         ld      (hl),a          ; scb_17
03A8' E5 40       and     01000000B      ; Test BIT 6
03AA' E5         push    hl
03AB' 2E B0       ld      1,scb_14
03AD' 01 0C70'   ld      bc,user
03B0' 54         ld      d,h
03B1' 1E ED       ld      e,scb_44
03B3' 1A         ld      a,(de)          ; Current User number
03B4' 02         ld      (bc),a
03B5' 7E         ld      a,(hl)          ; Default User number
03B6' C2 03BA'   jp      nz,cr_usr      ; If SCB_17 bit 6 set, set USER = current
03B9' 02         ld      (bc),a          ; else set USER = default user no
03BA' cr_usr:
03BA' 12         ld      (de),a          ; Set Current to Default User number
03BB' 03         inc     bc
03BC' 1E DA       ld      e,scb_3E
03BE' 1A         ld      a,(de)          ; Current Disk
03BF' C2 03C4'   jp      nz,cr_disk      ; SCB_17 bit 6 set set DISK = current
03C2' 3E FF       ld      a,0FFH         ; else to 0FFH
03C4' cr_disk:
03C4' 02         ld      (bc),a          ; Set DISK = current or 0FFH
03C5' 2B         dec     hl              ; SCB_13
03C6' 03         inc     bc
03C7' 7E         ld      a,(hl)
03C8' 02         ld      (bc),a          ; Set USE_DISK to default disk
03C9' 12         ld      (de),a          ; Set Current to Default Disk number

03CA' 2E EC       ld      1,scb_50
03CC' 03         inc     bc
03CD' 7E         ld      a,(hl)
03CE' 02         ld      (bc),a          ; Temporary disk (0 = default)

03CF' E1         pop     hl
03D0' 7E         ld      a,(hl)          ; scb_17
03D1' E5 80       and     10000000B      ; test for BDOS FUNCTION 47 - Program chain
03D3' CA 03E7'   jp      z,test_19      ; -no-

```

;

```

; BDOS FUNCTION 47 - Program chain
;-----
03D6' 21 0080          ld    hl,SET_CMD      ; Read CCP command from default buffer
03D9'                  copy_cmd:
03D9' 11 0CF5'          ld    de,c_buff+1
03DC' 0E 7F           ld    c,127           ; copy 127 bytes even though command terminate
03DE' 79              ld    a,c             ; by a NULL
03DF' 12              ld    (de),a
03E0' 13              inc    de
03E1' CD 0AAE'         call   copy_c         ; copy command line from buffer
03E4' C3 048B'         jp     command_line   ; and process

;-----
; Test for PROFILE.SUB
;-----
03E7'                  test_19:
03E7' 2E B5           ld    l,scb_19
03E9' 7E              ld    a,(hl)
03EA' E6 02           and    0000010B       ; test for first time load
03EC' C2 0406'        jp     nz,no_chain
03EF' 7E              ld    a,(hl) ; scb_19
03F0' F6 02           or     0000010B       ; set not first time load
03F2' 77              ld    (hl),a
03F3' 32 0C67'        ld    (sav_scb_19),a ; and save locally
03F6' 21 03FC'        ld    hl,prof_s
03F9' C3 03D9'        jp     copy_cmd

03FC' 50 52 4F 46     prof_s: DB    'PROFILE.S'
0400' 49 4C 45 2E
0404' 53
0405' 00              DB    000H           ; terminator

0406'                  no_chain:
0406' CD 0AE4'         quit:  call   set_scb_18 ; Return HL -> scb_18 after setting bits 5&7
0409' CD 0B09'         call   crlf

;-----
; Start of CCP-PROMPT procedure
; leading to > and request of CCP command
;-----
040C'                  ccp_prompt:
040C' 21 0E2B'        ld    hl,Top_sp-2
040F' F9              ld    sp,hl
0410' AF              xor    a
0411' 32 0C99'        ld    (y_curs),a
0414' 21 040C'        ld    hl,ccp_prompt
0417' E5              push   hl
0418' CD 0AE4'        call   set_scb_18 ; Return HL -> scb_18 after setting bits 5&7
041B' 2B              dec    hl
041C' 7E              ld    a,(hl) ; scb_17
041D' E6 01           and    1
041F' CA 0464'        jp     z,show_prompt

;-----
; $$$SUB file present as scb_17 bit 0 set ;
;-----
0422' 11 0CF5'        ld    de,c_buff+1
0425' CD 087B'        call   set_dra

```

```

0428' 0E 0F          ld    c,15          ; BDOS Open File
042A' CD 08F0'       call   BDOS_dol      ; NB FCB.CR is uninitialised
;*****
042D' 0E 0B          jp     nz,no_sub    ; -no file-
042F' CC 08F0'       ld     c,11         ; BDOS Get Console Status
0432' C2 044F'       call   z,BDOS_dol   ; (why not call bdos_tst)
0435' 21 0C96'       jp     nz,quit_sub  ; character entered or no file
0438' 77             ld     hl,sub_fcb + 35
0439' 2B             ld     (hl),a       ; R2=0
043A' 77             dec     hl          ;
043B' 2B             ld     (hl),a       ; R1=0
043C' E5             dec     hl          ;
043D' 3A 0C82'       push    hl
0440' 3D             ld     a,(sub_fcb + 15)
0441' 77             dec     a
0442' 77             ld     (hl),a       ; R0=RC-1
0444' 0E 21          ld     c,33         ; BDOS Read Random
0447' F4 08F0'       call   p,BDOS_dol   ; if RC > 0
0448' E1             pop     hl
0449' 35             dec     (hl)        ; R0 = R0 -1
044B' 0E 13          ld     c,19         ; BDOS Delete File
044E' FC 08F0'       call   m,BDOS_dol   ; if R0 < 0
044F' B7             or     a
044F' quit_sub:
044F' F5             push    af          ; ZF result of OPEN/READ or DELETE
0450' 0E 63          ld     c,99         ; BDOS Truncate File
0452' CD 08F0'       call   BDOS_dol     ; to R0,R1,R2
0455' F1             pop     af
0456' CA 04B8'       jp     z,command_line ; $$$SUB record read
0459' NO_SUB:
0459' 01 B301        ld     bc,scb_17 SHL 8 + 00000001B
045C' CD 0AF1'       call   clr_scb       ; clear $$$SUB flag
045F' 0E 13          ld     c,19         ; BDOS Delete File
0461' CD 08F0'       call   BDOS_dol     ; if it exists

;-----
; Display CCP prompt and wait for input
;-----
0464' show prompt:
0464' 3A 0C70'       ld     a,(user)
0467' B7             or     a
0468' C4 0B13'       call   nz,dsp_usr
046B' CD 056D'       call   vdu_cr_dsk
046E' 3E 3E          ld     a,'>'
0470' CD 0816'       call   vdu_out
0473' 11 B1BA        ld     de,scb_15 SHL 8 + scb_1E
0476' CD 0AA7'       call   copy2SCB     ; copy scb_15/16 to scb_1E/1F
0479' B7             or     a           ; test scb_16
047A' F5             push    af
047B' 01 B480        ld     bc,scb_18 SHL 8 + 10000000B
047E' C4 0AF1'       call   nz,clr_scb   ; clear scb_18 bit 7
0481' CD 084E'       call   get_con_buff
0484' CD 0AEE'       call   clr_scb_18   ; clear scb_18 bits 5&7
0487' F1             pop     af
0488' C4 09A8'       call   nz,CCP_RSX   ; multiple line is in RSX

;-----
; command line in c_buff ;

```

```

;-----;
048B'      command_line:
048B'      call   tst_scb_18      ; test SCB_18 bit 6
048E'      jp     nz,command_line1
0491'      2E C9      ld     l,scb_2D
0493'      7E      ld     a,(hl)
0494'      2B      dec     hl
0495'      77      ld     (hl),a      ; copy scb_2D to scb_2C
0496'      command_line1:
0496'      2E C8      ld     l,scb_2C
0498'      7E      ld     a,(hl)
0499'      32 0C9A'   ld     -(sav_scb_2C),a
049C'      CD 08F6'   call   scan_cop
049F'      C8      ret     z
04A0'      11 0CAC'   ld     de,fcblusr
04A3'      CD 0A39'   call   command

04A6'      3A 0CB6'   ld     a,(fcbl_t)
04A9'      FE 20      cp     ' '      ; is file TYPE a blank
04AB'      C2 0504'   jp     nz,go_command      ; -no-

04AE'      21 0CAC'   ld     hl,fcblusr
04B1'      7E      ld     a,(hl)      ; test if user number specified
04B2'      23      inc     hl
04B3'      B6      or     (hl)      ; or drive specified
04B4'      23      inc     hl
04B5'      7E      ld     a,(hl)      ; 1st character in file name
04B6'      C2 050A'   jp     nz,colon      ; -yes-

; no drive or user specified so scan BUILT IN commands first

04B9'      21 0537'   ld     hl,CCP_BI
04BC'      11 0CAE'   ld     de,fcbl_n
04BE'      3A 0CB0'   ld     a,(fcbl_n+2)      ; start of FILENAME
04C2'      FE 21      cp     ' '+1      ; 3rd character
04C4'      D4 0BFF'   call   nc,cmp_set      ; test for non blank
04C7'      C2 04E8'   jp     nz,command_type      ; -YES-
04CA'      3A 0CA1'   ld     a,(flag_5B)      ; less then 3 char or not B.I.
04CD'      B7      or     a      ; open square bracket flag
04CE'      78      ld     a,b      ; BI function found
04CF'      2A 0C6C'   ld     hl,(ptr_line)
04D2'      22 0C9D'   ld     (ptr_bi),hl
04D5'      21 055A'   ld     hl,BI_PROC
04D8'      CA 0758'   jp     z,jp_hl_a      ; no square bracket - execute B.I.F.
04DB'      FE 04      cp     4      ; test for DIR TYPE ERASE RENAME
04DD'      DA 0686'   jp     c,go_XBI      ; -yes- execute COM file
; Perhaps USER and DIRS may exist as PRL'S (from MP/M maybe)
04E0'      21 0CB1'   ld     hl,fcbl_n+3      ; 4th character in filename
; Test for DIRSYS built in function with square bracket
04E3'      C2 04E8'   jp     nz,command_type      ; -no- execute USER COM file
04E5'      36 20      ld     (hl),' '      ; set to blank for DIRS.S built in filename

;-----;
; Search for COM SUB or PRL file
;-----;
04E8'      command_type:
04E8'      01 B418   ld     bc,scb_18 SHL 8 + 00011000B

```

```

04EB'  CD 0ADD'          call    tst_scb
04EE'  CA 0504'          jp      z,go_command    ; if zero then use file type supplied
04F1'  06 08            ld      b,8
04F3'  90              sub     b
04F4'  CA 04F9'          jp      z,type_search    ; if 01B then try COM followed by SUB
04F7'  06 00            ld      b,0              ; else try SUB or FRL followed by COM
04F9'                                     type_search:
04F9'  C5              push    bc
04FA'  CD 077A'          call    search_type      ; use COM SUB or FRL for A = 0, 8 or 16
04FD'  CD 06EA'          call    load_tp          ; load transient prog and go to TPA if ok
0500'  F1              pop     af
0501'  CD 077A'          call    search_type      ; then try COM or SUB for B = 0 or 8
0504'                                     go_command:
0504'  CD 06EA'          call    load_tp          ; load transient prog and go to TPA if ok
0507'  C3 0B3A'          jp      tst_quit

;-----
; command contains user number and or drive name
;-----
050A'  FE 20            colon:  op      ' '          ; test if file name specified
050C'  C2 04EB'          jp      nz,command_type ; -yes-
050F'  CD 0B36'          call    tst_cmd_end      ; make sure no more in command line (unless SUB)
0512'  3A 0CAC'          ld      a,(fcb1usr)
0515'  D6 01            sub     1
0517'  DA 0525'          jp      c,new_dsk        ; -no-
051A'                                     new_usr:
051A'  32 0C70'          ld      (user),a
051D'  06 B0            ld      b,scb_14          ; Default User Number
051F'  CD 0AF9'          call    put_scb
0522'  CD 0886'          call    set_usr          ; Current User Number
0525'                                     new_dsk:
0525'  3A 0CAD'          ld      a,(fcb1)
0528'  3D              dec     a                  ; test for drive number specified
0529'  F8              ret     m                  ; -no-
052A'  F5              push    af
052B'  CD 0880'          call    login_dsk        ; BIOS select disk in <A>
052E'  F1              pop     af
052F'  32 0C72'          ld      (use_dsk),a
0532'  06 AF            ld      b,scb_13          ; default disk
0534'  C3 0AF9'          jp      put_scb

;-----
; CP/M 3 Built-in Commands
; (also COM filename except
; for DIRSYS which is DIRS)
;-----
0537'                                     CCP_BI:
0537'  44 49 52 20      db      'DIR '
053B'  54 59 50 45      db      'TYPE '
053F'  20
0540'  45 52 41 53      db      'ERASE '
0544'  45 20
0546'  52 45 4E 41      db      'RENAME '
054A'  4D 45 20
054D'  44 49 52 53      db      'DIRSYS '
0551'  59 53 20

```

```

0554' 55 53 45 52      db  'USER '
0558' 20
0559' 00              DB  000H

```

```

055A'                  BI_PROC:
055A' 0575'            DW  BI_DIR
055C' 05F4'            DW  BI_TYP
055E' 0622'            DW  BI_ERA
0560' 0651'            DW  BI_REN
0562' 057D'            DW  BI_IRS
0564' 0615'            DW  BI_USR

```

```

;
; Display logical disk name
; CMD_DSK - disk entered from command into zero page FCB (0 -> default)
; DSK      - disk supplied in A (0 -> default)
; CR_DSK   - default disk
;

```

```

0566'                  vdu_cmd_dsk:
0566' 3A 005C          ld    a,(SET_FCB)
0569'                  vdu_dsk:
0569' 3D              dec    a
056A' F2 0570'        jp     p,vdu_dsk1
056D'                  vdu_cr_dsk:
056D' 3A 0C72'        ld     a,(use_dsk)
0570'                  vdu_dsk1:
0570' C6 41          add    a,'A'
0572' C3 0BA6'        jp     vdu_a

```

```

;
; BUILT IN FUNCTION - DIR - List directory with DIR attribute
;

```

```

0575' 0E 00          BI_DIR: ld    c,0          ; t2' match (SYS attribute not set)
0577' 11 0C52'        ld     de,reg_sf
057A' C3 0582'        jp     BI_DS

```

```

;
; BUILT IN FUNCTION - DIRS - List directory with SYE attribute
;

```

```

057D' 0E 80          BI_IRS: ld    c,080H      ; t2' match (SYS attribute set)
057F' 11 0C4E'        ld     de,reg_ns
0582' D5              BI_DS: push  de
0583' CD 0599'        call  BI_DS1
0586' D1              pop    de
0587' CA 06B8'        jp     z,err_fn
058A' 7D              ld     a,l
058B' B8              cp     b
058C' D4 0B09'        call  nc,crlf
058F' 21 0C4D'        ld     hl,BI_D_FLAG      ; test flag
0592' 35              dec    (hl)
0593' 34              inc    (hl)
0594' C8              ret     z
0595' 35              dec    (hl)              ; clear flag
0596' C3 0B05'        jp     lineout          ; display message in DE

0599' C5              BI_DS1: push  bc

```



```

059A'  CD 0878'          call  set_def_dma
059D'  CD 09D8'          call  get_def_fcb      ; returns CF=Z if no FCB returned
05A0'  11 005D          ld     de,set_fcb+1    ; (DE already = 005CH)
05A3'  1A              ld     a,(de)
05A4'  FE 20           cp     ' '
05A6'  06 0B          ld     b,11
05A8'  CC 0BB5'        call  z,fill_3F
05AB'  CD 0B36'        call  tst_cmd_end      ; make sure no more in command line (unless SUB)
05AE'  CD 08C3'        call  search_first     ; DE -> FCB found
05B1'  C1             pop     bc
05B2'  C8             ret     z              ; no FCB found
05B3'  3A 0C97'        ld     a,(mod_scb_1A)   ; number of 16 byte fields in console width
05B6'  6F             ld     l,a
05B7'  47             ld     b,a
05B8'  04             inc     b
05B9'  E5             BI_DS2: push    hl
05BA'  21 000A          ld     hl,10          ; offset to T2'
05BD'  19             add     hl,de
05BE'  7E             ld     a,(hl)
05BF'  E1             pop     hl
05C0'  EB 80           and     080H          ; test for SYS file attribute
05C2'  B9             cp     c              ; against <> (DIR or DIRS)
05C3'  CA 05CE'        jp     z,BI_DS3       ; -yes-
05C6'  3E 01          ld     a,1
05C8'  32 0C4D'        ld     (BI_D_FLAG),a   ; set SYS/DIR FILES FOUND flag
05CB'  C3 05E5'        jp     BI_DS4

05CE'  05             BI_DS3: dec     b
05CF'  CC 0B08'        call  z,crlf_b        ; return with B = L
05D2'  78             ld     a,b
05D3'  BD             cp     l
05D4'  CC 0566'        call  z,vdw_cmd_dsk
05D7'  3E 3A          ld     a,':'
05D9'  CD 0BA6'        call  vdu_a
05DC'  CD 0BA4'        call  vdu_bl
05DF'  CD 0B8D'        call  vdu_fn
05E2'  CD 0BA4'        call  vdu_bl
05E5'  C5             BI_DS4: push    bc
05E6'  E5             push    hl
05E7'  CD 0866'        call  console_abort
05EA'  CD 08C3'        call  search_next
05ED'  E1             pop     hl
05EE'  C1             pop     bc
05EF'  C2 05B9'        jp     nz,BI_DS2
05F2'  3C             inc     a
05F3'  C9             ret

;
; BUILT IN FUNCTION - TYPE - Display file in ASCII on console
;
05F4'  21 0406'        BI_TYP: ld     hl,quit
05F7'  E5             push    hl
05F8'  CD 09D8'        call  get_def_fcb      ; returns DE -> SET_FCB
05FB'  3E 7F          ld     a,128-1        ; set pointer in record to E.O.R.
05FD'  32 0C9F'        ld     (bi_rec_ptr),a  ; (this forces read next record)
0600'  0E 0F          ld     c,15          ; BIOS Open File

```

```

0602'  CD 066C'          call    BI_bdos
0605'          BI_TYP1:
0605'          CD 0866'          call    console_abort
0608'  CD 0ACA'          call    type_char
060B'  CD              ret     nz
060C'  FE 1A            cp     01AH
060E'  C8              ret     z
060F'  CD 0816'          call    vdu_out
0612'  C3 0605'          jp     BI_TYP1

```

```

;
; BUILT IN FUNCTION - USER - Set user number (0-15)
;

```

```

0615'  11 0BF3'          BI_USR: ld     de,msg_us
0618'  CD 06AB'          call    get_tail      ; get command tail or prompt for it
061B'  CD 0B52'          call    validate_usr
061E'  C8              ret     z
061F'  C3 051A'          jp     new_usr

```

```

;
; BUILT IN FUNCTION - ERA - Erase file
;

```

```

0622'  CD 09D6'          BI_ERA: call    get_def_fcb
0625'  CA 064C'          jp     z,BI_ERA1      ; no fcb found
0628'  CD 069E'          call    tst_wildcard
062B'  C2 064C'          jp     nz,BI_ERA1      ;none found-
062E'  11 0C14'          ld     de,msg_er
0631'  CD 0849'          call    strout
0634'  2A 0C9B'          ld     hl,(ptr_tok)
0637'  0E 20            ld     c,' '           ; Blank terminates string
0639'  CD 0E2A'          call    vdumsg
063C'  11 0C1B'          ld     de,msg_yn
063F'  CD 0841'          call    prompt
0642'  CD 0B09'          call    crlf
0645'  7D              ld     a,l
0646'  B6 5F            and     01011111B      ; mask lower case
0648'  FE 59            cp     'Y'
064A'  C0              ret     nz             ; not 'Y' or 'y'
064B'  B7              or     a
064C'          BI_ERA1:
064C'          0E 13      ld     c,19           ; BDOS Delete File
064E'  C3 066C'          jp     BI_bdos

```

```

;
; BUILT IN FUNCTION - REN - Rename file
;

```

```

0651'  CD 09D6'          BI_REN: call    get_def_fcb      ; returns DE -> SET_FCB
0654'  F5              push    af
0655'  21 0010          ld     hl,16
0658'  19              add     hl,de
0659'  EB              ex      de,hl
065A'  D5              push    de           ; DE -> 2nd half of FCB
065B'  E5              push    hl           ; HL -> SET_FCB
065C'  0E 10            ld     c,16
065E'  CD 0AAE'          call    copy_c       ; copy 1st filename to 2nd half of FCB
0661'  CD 09D6'          call    get_def_fcb      ; fetch 2nd filename, returns DE = SET_FCB

```

```

0664' E1          pop    hl          ; HL -> SET_FCB (2nd filename)
0665' D1          pop    de          ; DE -> SET_FCB+16 (1st filename)
0666' CD 0692'    call    BI_REN_DSK
0669' 0E 17       ld      c,23      ; BDOS Rename File
066B' F1          pop    af
066C'             BI_bdos:          ; if ZF = 0, then no FCB passed
066C' F5          push   af
066D' C4 0B36'    call    nz,tst_cmd_end ; make sure no more in command line (unless SUB)
0670' F1          pop    af
0671' 11 005C     ld      de,SET_FCB
0674' 06 FF       ld      b,0FFH
0676' 26 01       ld      h,1
0678' C4 0891'    call    nz,fdos     ; C -> BDOS FNC, DE -> FCB
067B' C0          ret     nz          ; FCB entered, found and function completed
067C' 25          dec     h
067D' FA 06B8'    jp      m,err_fn
0680' 2A 0C3D'    ld      hl,(ptr_bi)
0683' 22 0C3C'    ld      (ptr_line),hl
;-----
; EXTENDED BUILT IN FUNCTION
; Load COM file for built in function
; as either BI command tail contains an open square bracket
; or BI function failed
;-----
0686'             go_XBI:
0686' CD 06EA'    call    load_tp      ; load transient prog and go to TPA if ok
0689' CD 0B8D'    call    vdu_fn
068C' 11 0C0A'    ld      de,msg_rq
068F' C3 06BB'    jp      err_msg
;-----
; Set disk for source and destination fcb for rename function
; On entry: DE -> destination filename, HL -> source filename
;-----
0692'             BI_REN_DSK:
0692' 1A          ld      a,(de)
0693' BE          cp      (hl)
0694' C8          ret     z           ; ok as both the same
0695' B7          or      a
0696' C8          ret     z           ; ok as destination set to default
0697' 34          inc     (hl)
0698' 35          dec     (hl)        ; test source for default
0699' C2 0B3A'    jp      nz,tst_quit ; bad as different and neither default
069C' 77          ld      (hl),a      ; set source to destination drive
069D' C9          ret
;-----
069E'             tst_wildcard:      ; search filename & type for wild card
069E' 06 0B       ld      b,8+3
06A0' 13          nxt_wc: inc     de      ; FCB+1
06A1' 1A          ld      a,(de)
06A2' FE 3F       cp      '?'
06A4' C8          ret     z
06A5' 05          dec     b
06A6' C2 06A0'    jp      nz,nxt_wc
06A9' 05          dec     b
06AA' C9          ret               ; return NZ for none

```

```

;
; BI_USER - Fetch command tail, or prompt for it if none entered
; On entry register DE -> prompt message
;
get_tail:
06AB'      call    ccp_ns      ; scan command for start of token
06AB'      CD 0941'
06AE'      ret     nz         ; -token found-
06AF'      call    strout
06B2'      CD 0849'
06B5'      call    get_con_buff
06B5'      C3 08F6'          jp     scan_ccp

06B8'      11 0C02'
06BB'      err_fn: ld     de,msg_nf
06BB'      err_msg:
06BB'      call    lineout
06BE'      C3 040C'          jp     ccp_prompt

06C1'      00
06C2'      53 55 42 4D
06C6'      49 54 20 20
06CA'      43 4F 4D

fn_sub: DB    000H
         DB    'SUBMIT COM'

```

```

;
; Insert SUBMIT.COM into CCP command to execute .SUB file
;
SUBMIT: ld    a,(de)
         ld    b,sch_of
06CD'      1A
06CE'      06 AB
06F0'      CD 0AF9'
06D3'      21 06C1'
06D6'      0E 0C
06D8'      CD 0AAE'
06DB'      21 0CF5'
06DE'      36 20
06E0'      23
06E1'      22 0C6C'
         ld    hl,c_buff+1
         ld    (hl),020H;' '
         inc   hl
         ld    (ptr_line),hl

```

```

;
; Search for COM SUB or PRL file on specified drive or through search chain
; If successful then load and go to TPA
;
load_tp:
06E4'      ld     de,fcbl_t      ; Source string
06E4'      11 0C86'
06E7'      ld     hl,com_sub_prl ; Compare with COM SUB PRL
06E7'      21 0763'
06EA'      call    cmp_set      ; return B = 0, 1 or 2 for COM SUB PRL
06ED'      ret     nz         ; not found
06ED'      00

06EE'      11 0CAC'
06F1'      ld     a,(de)
06F1'      1A
06F2'      or     a
06F3'      ret     nz

06F4'      inc    de
06F5'      ld     a,(de)
06F5'      1A
06F6'      ld     c,a
06F7'      push   bc
06F8'      ld     c,0          ; Set 0 drives to be search

```

```

06FA' B7          or      a
06FB' C2 0721'     jp      nz,l_tp4
                    ; search for file on up to 4 drives specified
06FE' 01 E704     ld      bc,+(scb_4C-1) SHL 8 + 04H      ; 4 Drive Search Chain
0701' 3A 0C72'     ld      a,(use_dsk)
0704' 3C          inc      a          ; A = drive + 1
0705' 67          ld      h,a
0706' 2E 01       ld      l,1
0708' 04          1_tp1: inc      b
0709' 0D          dec      c          ; decrement drives to be searched
070A' 79          ld      a,c
070B' E5          push     hl
070C' F4 0AFF'     call    p,get_scb
070F' E1          pop      hl
0710' B7          or      a
0711' FA 0776'     jp      m,l_tp6          ; not found as no more drives to search
0714' CA 071B'     jp      z,l_tp2
0717' BC          cp      h
0718' C2 0720'     jp      nz,l_tp3
071B' 7C          1_tp2: ld      a,h
071C' 2D          dec      l
071D' FA 0708'     jp      m,l_tp1
0720' 12          1_tp3: ld      (de),a

0721' C5          1_tp4: push     bc
0722' E5          push     hl
0723' CD 088B'     call     open_fcb1          ; Open file
0726' E1          pop      hl
0727' C1          pop      bc
0728' CA 0708'     jp      z,l_tp1

072B' 01 B403     ld      bc,scb_18 SHL 8 + 00000011B
072E' CD 0ADD'     call     tst_scb
0731' CA 0754'     jp      z,l_tp5 ; dont display file loaded
0734' 1A          ld      a,(de)          ; fcb1 dr
0735' CD 0569'     call     vdu_dsk
0738' 3E 3A       ld      a,':'
073A' CD 0BA6'     call     vdu_a
073D' D5          push     de
073E' CD 0B8D'     call     vdu_fn
0741' D1          pop      de
0742' D5          push     de
0743' 21 0008     ld      hl,8
0746' 19          add      hl,de
0747' 7E          ld      a,(hl)          ; f8
0748' E6 80       and      080H          ; test reserved flag
074A' 11 0C42'     ld      de,msg_u0
074D' C4 0849'     call     nz,strout          ; -set-
0750' CD 0B09'     call     crlf
0753' D1          pop      de
0754' F1          1_tp5: pop      af          ; recover B into A ( 0,1, or 2)
0755' 21 0770'     ld      hl,type_proc
                    ;-----
                    ; jump to A'th vector in table at HL
                    ;-----
0758'             jp hl,a:

```

```

0758' 87          add    a,a
0759' CD 0BB0'    call   a2hl
075C' D5          push   de
075D' 5E          ld     e,(hl)
075E' 23          inc    hl
075F' 56          ld     d,(hl)
0760' EB          ex     de,hl
0761' D1          pop    de
0762' E9          jp     (hl)

```

```

0763'                                com_sub_prl:
0763' 43 4F 4D 20          DB     'COM '
0767' 53 55 42 20          DB     'SUB '
076B' 50 52 4C 20          DB     'PRL '
076F' 00                  DB     000H

```

```

0770'                                type_proc:
0770' 0789'                DW     OVERLY
0772' 06CD'                DW     SUBMIT
0774' 0789'                DW     OVERLY

```

```

0776' C1            l_tp6: pop    bc
0777' 79            ld     a,c
0778' 12            ld     (de),a
0779' C9            ret

```

```

077A' 0F            search_type: rrca
077B' 21 0763'      ld     hl,com_sub_prl
077E' CD 0BB0'      call   a2hl
0781' 11 0CB6'      ld     de,fcbl_t
0784' 0E 03         ld     c,3
0786' C3 0AAE'      jp     copy_c

```

---

```

;
; USE BDOS LOAD OVERLAY to load and execute COM or PRL file
; On entry DE -> FCB
; NB this uses parts of loader area for copy of FCB and stack
;   FCB - page below BDOS + COH (35 bytes)
;   SP - TOP at base page of BDOS (space for 29 bytes)
;

```

---

```

0789' 21 0100      OVERLY: ld     hl,tpa
078C' 22 0CCE'      ld     (tp_base),hl

078F' 2A 0C9F'      ld     hl,(bdos_pg-1)
0792' 25          dec     h
0793' 2E C0        ld     l,0COH
0795' E5          push   hl           ; save address of FCB
0796' 1A          ld     a,(de)
0797' 32 0050      ld     (SET_DR),a
079A' EB          ex     de,hl
079B' 0E 23        ld     c,35
079D' CD 0AAE'      call   copy_c           ; copy FCB into top of loader

07A0' 21 0C67'      ld     hl,sav_scb_19
07A3' 3A          inc     (hl)

```

```

07A4' 2A 0C8C'      ld      hl,(ptr_line)
07A7' 2B            dec      hl
07AB' 11 0081      ld      de,SET_CMD+1
07AB' EB          ex       de,hl
07AC' 22 0C8C'      ld      (ptr_line),hl ; just in case it dont work
07AE' CD 0AB7'      call     copy_tail
07B2' 32 0080      ld      (SET_CMD),a ; length of command tail copied

07B5' CD 09D8'      call     get_def_fcb ; if password B > 0 & HL -> password
07B8' 22 0051      ld      (SET_PA1),hl ; save start of 1st password
07BB' 78          ld      a,b
07BC' 32 0053      ld      (SET_PL1),a ; set 1st password length

07BF' 11 006C      ld      de,SET_F2
07C2' CD 09DB'      call     get_fcb ; if password B > 0 & HL -> password
07C5' 22 0054      ld      (SET_PA2),hl ; save start of 2nd password
07C8' 78          ld      a,b
07C9' 32 0056      ld      (SET_PL2),a ; set 2nd password length

07CC' 21 0C71'      ld      hl,disk
07CE' 7E          ld      a,(hl)
07D0' B7          or       a
07D1' F4 0880'      call     p,login_dsk ; BDOS select disk in <A>

07D4' 3A 0C70'      ld      a,(user)
07D7' CD 0886'      call     set_usr
07DA' 87          add     a,a
07DB' 87          add     a,a
07DC' 87          add     a,a
07DD' 87          add     a,a
07DE' 2E DA        ld      l,scb_3E
07ED' B6          or       (hl)
07E1' 32 0004      ld      (set_dsk),a ; Set zero page DSK & USR

07EA' D1          pop     de ; restore address of FCB
;
; Set stack and initialise for execution starting at 0100H
; and for RET from TPA program to 0000H
;
07E5' 2A 0C9F'      ld      hl,(bdos_pg-1)
07E8' AF          xor      a
07E9' 6F          ld      l,a
07EA' F9          ld      sp,hl ; set stack to top of LOADER program
07EB' 67          ld      h,a
07EC' E5          push     hl ; top of stack = 0000H
07ED' 24          inc      h
07EE' E5          push     hl ; next entry = 0100H
07EF' 32 007C      ld      (set_fcb+32),a ; set CR to zero
07F2' 06 CF          ld      b,scb_33
07F4' CD 0AF9'      call     put_scb
07F7' 2E 90          ld      l,xscb ; 12th byte below SCB
07F9' 77          ld      (hl),a
07FA' 23          inc      hl
07FB' 77          ld      (hl),a ; 11th byte below SCB
07FC' 23          inc      hl
07FD' 77          ld      (hl),a ; 10th byte below SCB

```

```

07FE' 23          inc    hl
07FF' 77          ld      (hl),a      ; 9th byte below SCB

0800' 2E B3       ld      l,scb_17
0802' 7E          ld      a,(hl)
0803' E6 80       and     1000000B    ; Bit 7 - Also Set by BIOS FUNCTION 47
0805' C2 080D'    jp      nz,over11
0808' 2E AC       ld      l,scb_10    ; Program Error return code (2 bytes)
080A' 77          ld      (hl),a      ; set to zero
080B' 23          inc     hl
080C' 77          ld      (hl),a
080D' 7E          over11: ld      a,(hl)
080E' E6 3F       and     not 1100000b
0810' 77          ld      (hl),a
0811' 0E 3B       ld      c,59        ; Load Overlay (DE -> FCB)
0813' C3 0005     jp      BIOS        ; and start execution at TPA

```

```

;-----
; output to VDU of character in register A
;-----

```

```

0816'          vdu_out:          ; Output to screen
0816' FE 0A       cp      00AH        ; test for line feed
0818' C2 083B'    jp      nz,a_out
081B' 21 0C98'    ld      hl,sav_scb_1C ; YES test for bottom of screen
081E' 7E          ld      a,(hl)
081F' 23          inc     hl
0820' 34          inc     (hl)
0821' 96          sub     (hl)
0822' C2 0839'    jp      nz,lf_out
0825' 77          ld      (hl),a
0826' 23          inc     hl
0827' 7E          ld      a,(hl)
0828' B7          or      a
0829' 11 0C24'    ld      de,msg_rt    ; Press RETURN to continue
082C' CC 0841'    call    z,prompt
082F' FE 03       cp      3           ; test for CTRL-C
0831' CA 0406'    jp      z,quit      ; yes - QUIT
0834' 1E 0D       ld      e,cr
0836' CD 083C'    call    conout
0839' 3E 0A       lf_out: ld      a,lf
083B' 5F          a_out: ld      e,a
083C' 0E 02       conout: ld      c,2   ; Console output
083E' C3 0005     jp      BIOS

```

```

0841' CD 0849'    prompt: call    strout
0844' 0E 01       ld      c,1         ; Console Input
0846' C3 0005     jp      BIOS

```

```

0849' 0E 09       strout: ld      c,9   ; Print String
084B' C3 0005     jp      BIOS

```

```

084E'          get_con_buff:
084E' 21 0CF4'    ld      hl,c_buff
0851' 36 E7       ld      (hl),0E7H    ; (maximum length)
0853' EB          ex      de,hl
0854' 0E 0A       ld      c,10        ; Read Console Buffer

```



```

0856' 0D 0005      call  BDOS
0859' 21 0CF5'      ld    hl,c_buff+1
085C' 7E           ld    a,(hl)
085D' 23           inc    hl
085E' 0D 08B0'      call  a2hl
0861' 36 00        ld    (hl),0      ; add NULL to end of string
0863' C3 0B09'      jp     crlf

0866'              console_abort:
0866' 0D 086D'      call  conin      ; see if anything typed at console
0869' C8           ret             ; -no-
086A' C3 0406'      jp     quit     ; -yes- so abort built in function

086D'              conin:
086D' 0E 0B        ld    c,11      ; BDOS Console Status
086F' 0D 08EB'      call  bdos_tst  ; Call BDOS and TST result
0872' C8           ret             ; -Not ready-
0873' 0E 01        ld    c,1      ; BDOS Console Input
0875' C3 08EB'      jp     bdos_tst ; Call BDOS and TST result

0878'              set_def_dma:
0878' 11 0080      ld     de,SET_CMD
087B'              set_dma:
087B' 0E 1A        ld     c,26      ; Set DMA address
087D' C3 0005      jp     BDOS

0880'              login_dsk:
0880' 5F           ld     e,a
0881' 0E 0E        ld     c,14      ; Select disk in E
0883' C3 0005      jp     BDOS

0886'              set_usr:
0886' 06 ED        ld     b,scb_44
0888' C3 0AF9'      jp     put_scb

088B'              open_fcb1:
088B' 01 000F      ld     bc,15      ; BDOS: Open File
088E' 11 0CAD'      ld     de,fcb1
0891' 21 0020      fdos: ld     hl,32      ; offset to CR
0894' 19          add     hl,de
0895' 36 00        ld     (hl),0
0897' C5          push    bc
0898' D5          push    de
0899' 1A          ld     a,(de)
089A' A0          and     b
089B' 3D          dec     a
089C' F4 0880'      call  p,login_dsk ; BDOS select disk in <A>
089F' 11 0CA2'      ld     de,psword
08A2' 0D 087B'      call  set_dma
08A5' D1          pop     de
08A6' C1          pop     bc
08A7' D5          push    de
08A8' 2A 028D'      ld     hl,(PTR_SCB)
08AB' 2E E7        ld     l,scb_4B
08AD' 70          ld     (hl),b
08AE' E5          push    hl

```

```

08AF' CD 0005          call BDOS
08B2' D1               pop de
08B3' AF              xor a
08B4' 12              ld (de),a
08B5' 3A 0C72'        ld a,(use_dsk)
08B8' 1E DA           ld e,scb_3E
08BA' 12              ld (de),a
08BB' E5              push hl
08BC' CD 0878'        call set_def_dma
08BF' E1              pop hl
08C0' 2C              inc l
08C1' D1              pop de
08C2' C9              ret

08C3'                search_first:
08C3' 0E 11           ld c,17          ; Search for first
08C5' C3 08CA'        jp search_fcb
08C8'                search_next:
08C8' 0E 12           ld c,18          ; Search for next
08CA'                search_fcb:
08CA' 11 005C         ld de,SET_FCB
08CD' CD 0005         call BDOS          ; Search first/next
08D0' 3C              inc a
08D1' C8              ret z            ; -no (more) fcb found
08D2' 3D              dec a
08D3' 87              add a,a
08D4' 87              add a,a
08D5' 87              add a,a
08D6' 87              add a,a
08D7' 87              add a,a
08D8' 21 0080         ld hl,SET_CMD      ; default DMA area
08DB' CD 08B0'        call a2hl
08DE' EB              ex de,hl
08DF' AF              xor a
08E0' 3D              dec a
08E1' C9              ret                ; return DE -> FCB found, A > 0

08E2' AF              bi_read:          xor a
08E3' 32 0C9F'        ld (bi_rec_ptr),a
08E5' 0E 14           ld c,20          ; Read Sequential
08E8' 11 005C         ld de,SET_FCB
08EB'                bdos_tst:
08EB' CD 0005         call BDOS          ; Call BDOS and TST result
08EE' B7              or a
08EF' C9              ret

08F0'                BDOS_dol:
08F0' 11 0C73'        ld de,sub_fcb
08F3' C3 08EB'        jp bdos_tst       ; Call BDOS and TST result

;
;Scan CCP Command Line for comments, multiple command and blanks
;

08F6'                scan_ccp:
08F6' CD 0AEE'        call clr_scb_18    ; and return <HL> -> byte
08F9' EB              ex de,hl

```

```

08FA' AF          xor    a
08FB' 32 0CA1'     ld     (flag_5B),a
08FE' 21 0CF6'     ld     hl,c_buff+2
0901' CD 0944'     call   nxt_ns ; return HL -> to NULL or non blank/tab character
0904' EB          ex     de,hl
0905' FE 3B       cp     ',' ; test for comment
0907' C8          ret     z
0908' FE 21       cp     021H ; test for multiple command as first character
090A' CA 091C'     jp     z,skip_c
090D' FE 3A       cp     ':' ; test for colon as first character
090F' C2 091D'     jp     nz,chr_ok
0912' 2E AC       ld     l,scb_10
0914' 34          inc     (hl)
0915' 34          inc     (hl)
0916' CA 091C'     jp     z,skip_c ; (+scb_10) was = 0FEH
0919' 23          inc     hl
091A' 34          inc     (hl) ; test HIGH scb_10 (error return code)
091B' C8          ret     z
091C' 13          skip_c: inc de ; skip to next character in CCP command
091D' EB          chr_ok: ex de,hl
091E' 22 0C8C'     ld     (ptr_line),hl ; and save
0921'             nxt_occup:
0921' 7E          ld     a,(hl)
0922' FE 5B       cp     '[' ; Test for [
0924' C2 092A'     jp     nz,not_5B
0927' 32 0CA1'     ld     (flag_5B),a ; YES so set flag
092A' FE 61       not_5B: cp 'a'
092C' DA 0937'     jp     c,not_lc
092F' FE 7B       cp     'z'+1
0931' D2 0937'     jp     nc,not_lc
0934' D6 20       sub     'a'-'A' ; Convert LCASE to UCASE
0936' 77          ld     (hl),a
0937' FE 21       not_lc: cp 021H ; explanation mark
0939' CC 0959'     call   z,multiple
093C' 23          inc     hl
093D' B7          or     a
093E' C2 0921'     jp     nz,nxt_occup
;
; Move command line pointers passed any spaces or tabs
; Return Z if end of command NULL found
; Return NZ if character found and set ptr_line and ptr_tok to this character
; Registers DE and BC unchanged
;
0941'             ocp_ns:
0941' 2A 0C8C'     ld     hl,(ptr_line) ; search for non space or tab
0944' 22 0C8C'     nxt_ns: ld (ptr_line),hl
0947' 22 0C3B'     ld     (ptr_tok),hl
094A' 7E          ld     a,(hl)
094B' B7          or     a
094C' C8          ret     z ; return Z if NULL found
094D' FE 20       cp     ','
094E' CA 0955'     jp     z,skip_b
0952' FE 09       cp     9
0954' C0          ret     nz ; return NZ if any other character found
0955' 23          skip_b: inc hl
0956' C3 0944'     jp     nxt_ns

```

```

;-----
; CCP command line contains an !
;-----
0959' multiple:
0959' 5D      ld      e,l
095A' 54      ld      d,h
095B' 13      inc     de
095C' 1A      ld      a,(de)      ; test next character for !
095D' FE 21   cp      021H
095E' F5      push    af
0960' E5      push    hl
0961' CC 0AB7' call    z,copy_tail    ; -yes-
0964' E1      pop     hl
0965' F1      pop     af
0966' C8      ret      z          ; so quit
0967' 36 00   ld      (hl),0      ; make this location into a NULL
0969' EB      ex      de,hl

;-----
; Create an RSX in the page below BDOS ;
;-----
096A' 2A 0006      ld      hl,(BDOS+1)
096D' 25      dec     h
096E' 2E 18      ld      l,018H      ; 'loader' flag (00 for RSX or FF for loader)
0970' 77      ld      (hl),a      ; but first character goes in here
0971' E5      push    hl
0972' 23      m_rsx1: inc     hl
0973' 13      inc     de
0974' 1A      ld      a,(de)
0975' 77      ld      (hl),a
0976' FE 21   cp      021H
0978' C2 097D' jp      nz,m_rsx2
097B' 36 0D      ld      (hl),00DH      ; replace any further ! by <RETURN>
097D' B7      m_rsx2: or      a
097E' C2 0972' jp      nz,m_rsx1      ; loop till NULL
0981' 36 0D      ld      (hl),00DH      ; insert <RETURN>
0983' 23      inc     hl
0984' 77      ld      (hl),a      ; before NULL

0985' 2E 06      ld      l,6
0987' 36 C3      ld      (hl),0C3H
0989' 23      inc     hl
098A' 36 09      ld      (hl),9
098C' 23      inc     hl
098D' 74      ld      (hl),h      ; initialise RSX+06 to JUMP RSX+09
098E' 23      inc     hl
098F' 36 C3      ld      (hl),0C3H      ; initialise RSX+09 to JUMP
0991' 2E 0E      ld      l,00EH
0993' 77      ld      (hl),a      ; initialise RSX+0E to 0 (REMOVE = false)
0994' 6F      ld      l,a
0995' EB      ex      de,hl
0996' CD 00D0' call    init_rsx      ; initialise as for standard RSX
0999' 2A 028D' ld      hl,(PTR_SCB)
099C' 2E B1      ld      l,scb_15
099E' D1      pop     de
099F' 13      inc     de      ; start at RSX+19

```

```

09A0' 73          ld  (hl),e
09A1' 23          inc  hl
09A2' 72          ld  (hl),d          ; set word at scb_15 to start of RSX line
09A3' 2E AE       ld  l,scb_12       ; and set scb_12 flag non-zero
09A5' 72          ld  (hl),d
09A6' AF         xor  a
09A7' C9         ret

```

```

;-----;
; Multiple line RSX existed prior to read of command line ;
;-----;

```

```

09A8' OCP_RSX:          ; (on entry H -> page of SCB)
09A8' 11 BAB1         ld  de,scb_1E SHL 8 + scb_15
09AB' CD 0AA7'       call copy2SCB      ; Copy scb_1E/1F back to scb_15/16
09AE' B7             or  a              ; test scb_1F
09AF' 11 BCB1         ld  de,scb_20 SHL 8 + scb_15
09B2' CC 0AA7'       call z,copy2SCB    ; -empty so copy scb_20/21 to scb_15/16
09B5' E5             push hl
09B6' CD 086D'       call conin         ; and set nz if input
09B9' E1             pop  hl
09BA' 2E B1          ld  l,scb_15
09BC' C2 09CB'       jp  nz,era_cmd_rsx ; -input-
09BE' 5E             ld  e,(hl)         ; LOW scb_15
09C0' 23             inc  hl
09C1' 56             ld  d,(hl)         ; HIGH scb_15
09C2' 34             inc  (hl)
09C3' 35             dec  (hl)         ; test for ZERO HIGH page
09C4' 2B             dec  hl
09C5' CA 09CB'       jp  z,era_cmd_rsx
09C8' 1A             ld  a,(de)         ; start of multiple line command
09C9' B7             or  a              ; test for NULL
09CA' C0             ret  nz           ; -no so ok-
09CB' era_cmd_rsx:    ; delete RSX
09CB'               xor  a
09CC' 77             ld  (hl),a         ; LOW scb_15
09CD' 23             inc  hl
09CE' 77             ld  (hl),a         ; HIGH scb_15

09CF' 2E AE          ld  l,scb_12       ; find address of scb
09D1' 66             ld  h,(hl)         ; page address
09D2' 2E 0E          ld  l,00EH        ; offset to REMOVE flag
09D4' 35             dec  (hl)         ; set TRUE
09D5' C3 0100'       jp  era_rsx       ; and erase

```

```

09D8' get_def_fcb:
09D8' 11 005C        ld  de,SET_FCB
09DB' get_fcb:
09DB'               call  ocp_ns        ; scan command for start of token
09DE' F5             push  af
09DF' CD 09E4'       call  parse_fcb
09E2'               pop  af
09E3' C9             ret              ; ZF = true if no token found

```

```

;-----;
; parse filename and get result
; On entry:  DE -> FCB

```

```

; HL -> start of CCP command token
; Returns: B -> length of password or zero if none
; HL -> start of password
; DE -> FCB
;
parse_fcb:
09E4'      ld      (ptr_line),hl      ; (CCP_NS also does this)
09E7'      ld      (ptr_tok),hl      ; (CCP_NS also does this)
09EA'      push    de
09EB'      ld      de,pfcb152
09EE'      ld      c,152              ; Parse Filename
09F0'      call    BDCS              ; into P_FCB
09F3'      pop     de
09F4'      ld      a,h
09F5'      or      l                  ; test for zero
09F6'      ld      b,(hl)             ; B = delimiter (or 0CDH for 0000)
09F7'      inc     hl
09F8'      jp      nz,parse1          ; NOT ZERO
09FB'      ld      hl,null_tok        ; as ZERO set delimiter address to null_tok
09FE'      ld      a,h
parse1:    ld      a,h
09FF'      or      l                  ; test for 0FFFFH
0A00'      jp      nz,parse2          ; -NO-
0A03'      ld      hl,null_tok        ; as ERROR set delimiter address to null_tok
0A06'      call    tst_quit           ; quit if (sav_scb_19) zero
0A09'      ld      a,b
parse2:    ld      a,b
0A0A'      cp      ','               ; test for POINT delimiter
0A0C'      jp      nz,not_fs          ; -NO-
0A0F'      dec     hl                 ; yes- so go back one character
0A10'      ld      (ptr_line),hl      ; update start of remainder of line
0A13'      ld      c,16
0A15'      ld      hl,P_FCB           ; FCB created
0A18'      push    de
0A19'      call    copy_c             ; copy 16 bytes to (DE)
0A1C'      ld      de,psword          ; (password is only 8 bytes)
0A1F'      ld      c,10              ; (next 2 bytes are reserved and unknown)
0A21'      call    copy_c             ; and copy next 10 bytes to psword
0A24'      pop     de
0A25'      ld      a,(hl)             ; reserved area - length of password
0A26'      ld      hl,0
0A28'      null_tok EQU $-1           ; default delimiter address -> 0 (naughty code)
0A29'      or      a                  ; test reserved byte 26 of P_FCB
0A2A'      ld      b,a                ; saved byte from reserved area
0A2B'      jp      z,parse4
0A2E'      ld      hl,(ptr_tok)
0A31'      ld      a,(hl)
parse3:    ld      a,(hl)
0A32'      cp      ','               ; scan field for password
0A34'      inc     hl
0A35'      jp      nz,parse3          ; HL -> start of password
0A38'      parse4: ret

;
; extract COMMAND from command line and file FCB1
;
command:
0A39'      push    de
0A3A'      xor     a

```

```

0A3B' 12          ld      (de),a          ; initialise USER to default
0A3C' 13          inc     de
0A3D' 12          ld      (de),a          ; initialise DR to default
0A3E' 13          inc     de
0A3F' CD 0941'    call    ocp_ns          ; scan command for start of token
0A42' 2A 0C8C'    ld      hl,(ptr_line)
0A45' D1          pop     de
0A46' D5          push    de

; Test first four characters searching for a colon
; stop this search when space or tab found
; abort this search if < 020h found

0A47' 06 04          ld      b,4
0A49' 7E          c_1st4: ld      a,(hl)
0A4A' FE 3A          cp      ':'
0A4C' CA 0A67'      jp      z,c_colon
0A4E' B7          or      a
0A50' C3 030A'      jp      patch
0A53' 05          c_not4: dec     b
0A54' 23          inc     hl
0A55' C2 0A49'      jp      nz,c_1st4

0A58' D1          c_not4: pop     de
0A59' AF          xor      a
0A5A' 12          ld      (de),a
0A5B' 2A 0C8C'    ld      hl,(ptr_line)
0A5E' 13          c_name: inc     de
0A5F' 1A          ld      a,(de)          ; FCB DR
0A60' F5          push    af
0A61' CD 09E4'    call    parse_fcb      ; If password B > 0, & HL -> start of password
0A64' F1          pop     af
0A65' 12          ld      (de),a          ; restore FCB DR
0A66' C9          ret

0A67' 2A 0C8C'    c_colon: ld      hl,(ptr_line)
0A6A' 7E          ld      a,(hl)
0A6B' FE 30          c_col1: cp      '0'
0A6D' DA 0A85'      jp      c,c_col2
0A70' FE 3A          cp      '9'+1
0A72' D2 0A85'      jp      nc,c_col2
0A75' CD 0B71'    call    asc2bin
0A78' D1          pop     de
0A79' D5          push    de
0A7A' 1A          ld      a,(de)
0A7B' B7          or      a
0A7C' C2 0A58'      jp      nz,c_not4
0A7E' 78          ld      a,b
0A80' 3C          inc     a
0A81' 12          ld      (de),a
0A82' C3 0A9C'      jp      c_col3
0A85' FE 41          c_col2: cp      'A'
0A87' DA 0A58'      jp      c,c_not4
0A8A' FE 51          cp      'P'+1
0A8C' D2 0A58'      jp      nc,c_not4
0A8E' D1          pop     de
0A90' D5          push    de
0A91' 13          inc     de

```

```

0A92' 1A          ld      a,(de)
0A93' B7          or      a
0A94' C2 0A58'    jp      nz,c_not4
0A97' 7E          ld      a,(hl)
0A98' D6 40       sub     'A'-1
0A9A' 12          ld      (de),a
0A9B' 23          inc     hl
0A9C' 7E          c_col3: ld      a,(hl)
0A9D' FE 3A       cp      ':'
0A9F' C2 0A6B'    jp      nz,c_col1
0AA2' 23          inc     hl
0AA3' D1          pop     de
0AA4' C3 0A5E'    jp      c_name

0AA7'             copy2SCB:             ; Copy 2 bytes of SCB from offset D to offset E
0AA7' 2A 028D'    ld      hl,(PIR_SCB)
0AAA' 6A          ld      l,d
0AAB' 54          ld      d,h
0AAC' 0E 02       ld      c,2
0AAE'             copy_c:             ; copy C bytes from (hl) to (de)
0AAE' 7E          ld      a,(hl)
0AAF' 12          ld      (de),a
0AB0' 23          inc     hl
0AB1' 13          inc     de
0AB2' 0D          dec     c
0AB3' C2 0AAE'    jp      nz,copy_c
0AB6' C9          ret

0AB7'             copy_tail:          .
0AB7' 0E 00       ld      c,0          ; Copy from (DE) to (HL)
0AB9'             next_copy:
0AB9' 1A          ld      a,(de)
0ABA' 77          ld      (hl),a
0ABB' B7          or      a          ; Stop after (DE) = 0
0ABC' 79          ld      a,c          ; Return length of copy in C
0ABD' C8          ret      z
0ABE' 23          inc     hl
0ABF' 13          inc     de
0AC0' 03          inc     bc
0AC1' C3 0AB9'    jp      next_copy

;-----
; Return next character from file
;-----
0AC4' AF          type_char:          xor      a
0AC5' 21 0C9F'    ld      hl,bi_rec_ptr
0AC8' 34          inc     (hl)          ; test 07FH or greater
0AC9' FC 08E2'    call    m,bi_read      ; after open do a read and set flag = 0
0ACC' B7          or      a
0ACD' C0          ret      nz
0ACE' 3A 0C9F'    ld      a,(bi_rec_ptr)
0AD1' 21 0080     ld      hl,SET_CMD
0AD4' CD 0BB0'    call    a2hl
0AD7' AF          xor      a
0AD8' 7E          ld      a,(hl)          ; return next character (ZF=1)
0AD9' C9          ret

```



```

OADA'          tst_scb_18:          ; test SCB_18 bit 6
OADA' 01 B440      ld      bc,scb_18 SHL 8+01000000b
OADD'          tst_scb:
OADD' 2A 028D'      ld      hl,(PTR_SCB)
OAE0' 68          ld      l,b
OAE1' 7E          ld      a,(hl)
OAE2' A1          and     c
OAE3' C9          ret

OAE4'          set_scb_18:          ; Set bits 5&7 in scb_18
OAE4' 01 B4A0      ld      bc,scb_18 SHL 8+01000000b      ; set bits
OAE7' CD 0ADD'      call    tst_scb
OAEA' 79          ld      a,c
OAE8' B6          or      (hl)
OAE9' 77          ld      (hl),a
OAE0' C9          ret

OAEF'          clr_scb_18:          ; clear bits 5&7 in scb_18
OAEF' 01 B4A0      ld      bc,scb_18 SHL 8+01000000b
OAF1'          clr_scb:
OAF1' CD 0ADD'      call    tst_scb
OAF4' 79          ld      a,c
OAF5' 2F          cpl
OAF6' A6          and     (hl)
OAF7' 77          ld      (hl),a
OAF8' C9          ret

;-----
; Put <A> into SCB byte at low (B)
;-----
OAF9'          put_scb:
OAF9' 2A 028D'      ld      hl,(PTR_SCB)
OAF0' 68          ld      l,b
OAF1' 77          ld      (hl),a
OAF2' C9          ret

;-----
; Get SCB byte at low (B) into <A>
;-----
OAF3'          get_scb:
OAF3' 2A 028D'      ld      hl,(PTR_SCB)
OAF4' 68          ld      l,b
OAF5' 7E          ld      a,(hl)
OAF6' C9          ret

;-----
; Display string in <DE> followed by cr & lf, return with B = L
;-----
OB05'          lineout:
OB05' CD 0849'      call    strout
OB08' 45          crlf_b: ld      b,l
OB09' 3E 0D          crlf:  ld      a,cr
OB0B' CD 0B46'      call    vdu_a
OB0E' 3E 0A          ld      a,lf
OB10' C3 0B46'      jp      vdu_a

```

```

;
; Display User number in decimal with one or two digits
;
OB13'      dsp_usr:
OB13'      sub    10          ; Display Number in <A>
OB15'      jp     c,dsp_dig
OB18'      ld     e,'0'
OB1A'
OB1A'      dsp_ten:
OB1A'      inc     e
OB1B'      sub    10
OB1D'      jp     nc,dsp_ten
OB20'      push   af
OB21'      call   conout
OB24'      pop    af
OB25'      dsp_dig:
OB25'      add    a,'0'+10
OB27'      jp     vdu_out

;
; Display message at HL on screen up to NULL terminator or = C terminator
;
OB2A'      vdumsg: ld    a,(hl)
OB2B'      or     a
OB2C'      ret    z
OB2D'      cp     c
OB2E'      ret    z
OB2F'      call   vdu_a
OB32'      inc    hl
OB33'      jp     vdumsg

;
; Test for any more tokens from command line
; Return if no more, or display the next token followed by '?' and quit
; Special case: if no more and sav_scb_19 > 0 then return even if more
; but with CF = NZ (scb_19 is submit/config flag)
;
OB36'      tst_cmd_end:
OB36'      call   ccp_ns ; scan command for start of token
OB39'      ret    z      ; -no token found-
OB3A'
OB3A'      tst_quit:
OB3A'      ld     hl,sav_scb_19
OB3D'      ld     a,(hl)
OB3E'      or     a
OB3F'      ld     (hl),0 ; set (sav_scb_19) zero for next time
OB41'      ret    nz      ; (sav_scb_19) not zero
OB42'      ld     hl,(ptr_tok)
OB45'      ld     c,' ' ; Blank terminates string
OB47'      call   vdumsg
OB4A'      ld     a,'?'
OB4C'      call   vdu_out
OB4F'      jp     quit

;
; extract user number from command line and
; validate numeric string in range 0-15

```

```

;-----
OB52' validate_usr:
OB52'      CD 0941'      call ccp_ns      ; scan command for start of token
OB55'      2A 0C3C'      ld hl,(ptr_line)
OB58'      22 0C3B'      ld (ptr_tok),hl    ; (this is already done in ccp_ns)
OB5B'      C8           ret z              ; -no token found-
OB5C'      7E           ld a,(hl)
OB5D'      FE 30       cp '0'
OB5F'      DA 0B3A'     jp c,tst_quit
OB62'      FE 3A       cp '9'+1
OB64'      D2 0B3A'     jp nc,tst_quit
OB67'      CD 0B71'     call asc2bin
OB6A'      22 0C3C'     ld (ptr_line),hl    ; update pointer to first non numeric
OB6D'      F6 01       or 1
OB6F'      78           ld a,b
OB70'      C9           ret

```

```

;-----
; Convert ASCII input in (HL) into binary number in <B>
; string terminated by first character outside range 0-9
; if binary number exceeds 15 - then error
;-----

```

```

OB71' asc2bin:
OB71'      06 00       ld b,0
OB73'      7E           ld a,(hl)
OB74'      D6 30       sub '0'
OB76'      D8           ret c
OB77'      FE 0A       cp 9+1
OB79'      D0           ret nc
OB7A'      F5           push af
OB7B'      78           ld a,b
OB7C'      87           add a,a
OB7D'      87           add a,a
OB7E'      80           add a,b
OB7F'      87           add a,a
OB80'      47           ld b,a      ; b = b * 10
OB81'      F1           pop af
OB82'      23           inc hl
OB83'      80           add a,b
OB84'      47           ld b,a
OB85'      FE 10       cp 15+1
OB87'      DA 0B73'     jp c,next_no
OB8A'      C3 0B3A'     jp tst_quit

OB8D'      13           vdu_fn: inc de
OB8E'      26 08       ld h,8
OB90'      CD 0B98'     call vdu_de
OB93'      CD 0BA4'     call vdu_bl
OB96'      26 03       ld h,3
OB98'      1A           vdu_de: ld a,(de)
OB99'      E6 7F       and 07FH      ; mask parity
OB9B'      CD 0BA6'     call vdu_a
OB9E'      13           inc de
OB9F'      25           dec h
OBA0'      C2 0B98'     jp nz,vdu_de
OBA3'      C9           ret

```

```

OBA4' 3E 20      vdu_bl: ld    a,' '
OBA6' C5         vdu_a: push  bc
OBA7' D5         push  de
OBA8' E5         push  hl
OBA9' CD 0816'   call   vdu_out
OBAC' E1         pop    hl
OBAD' D1         pop    de
OBAE' C1         pop    bc
OBAF' C9         ret

OBB0' 85         a2hl: add    a,l          ; hl = hl + a
OBB1' 6F         ld      l,a
OBB2' D0         ret      nc
OBB3' 24         inc     h
OBB4' C9         ret

OBB5'           fill_3F:
OBB5' 3E 3F      ld      a,03FH;'?'
OBB7' 12         nxt_3F: ld    (de),a
OBB8' 13         inc     de
OBB9' 05         dec     b
OBBA' C2 0BB7'   jp      nz,nxt_3F
OBBD' B7         or      a
OBBE' C9         ret

;
; Compare filename in DE against HL -> multiple matching strings
; Returns:      ZF = 1 if match found
;              A = pattern number found
; Note: source string may contain blanks (or less) which are not matched
;
OBBF'           cmp_set:
OBBF' 01 00FF   ld      bc,255          ; set C to -1
OBC2'           cmp_set1:
OBC2' D5         push    de
OBC3' E5         push    hl
OBC4'           cmp_set2:
OBC4' 1A         ld      a,(de)
OBC5' E6 7F     and     07FH          ; mask parity
OBC7' FE 21     cp      ' ' + 1      ; test for blank (or less)
OBC9' DA 0BD0'   jp      c,cmp_set3    ; -yes-
OBCB' BE         cp      (hl)         ; test against patterns
OBCD' C2 0BED'   jp      nz,cmp_set4   ; -no-
OBD0'           cmp_set3:
OBD0' 13         inc     de
OBD1' 0C         inc     c
OBD2' 3E 20     ld      a,' '          ; pattern delimiter
OBD4' BE         cp      (hl)         ; test end of pattern
OBD5' 23         inc     hl
OBD6' C2 0BC4'   jp      nz,cmp_set2   ; -no-
OBD9' E1         pop     hl
OBDA' D1         pop     de
OBD8' CD 0AAE'   call    copy_c        ; copy pattern to
OBDE' 78         ld      a,b          ; return A ->
OBEF' C9         ret

```

```

OBE0'                                cmp_set4:
OBE0' 3E 20                          ld    a,' '      ; pattern delimiter
OBE2'                                cmp_set5:
OBE2' BE                             cp    (hl)      ; test end of pattern
OBE3' 23                             inc    hl
OBE4' C2 OBE2'                       jp    nz,cmp_set5 ; -no- loop till found
OBE7' D1                             pop     de      ; (drop HL on stack)
OBE8' D1                             pop     de
OBE9' 04                             inc     b
OBEA' 0E FF                          ld     c,-1
OBEA' 7E                             ld     a,(hl)    ; test end of all patterns
OBE0' D6 01                          sub     1
OBEF' D2 OBC2'                       jp     nc,cmp_set1
OBF2' C3                             ret              ; -yes- return ZF = 0, CF = 1

OBF3' 45 6E 74 65                    msg_us:      db    'Enter User £: $'
OBF7' 72 20 55 73
OBF8' 65 72 20 23
OBF9' 3A 20 24
OC02' 4E 6F 20 46                    msg_nf:      db    'No File$'
OC06' 69 6C 65 24
OC0A' 20 72 65 71                    msg_rq:      db    ' required$'
OC0E' 75 69 72 65
OC12' 64 24
OC14' 45 52 41 53                    msg_er:      db    'ERASE $'
OC18' 45 20 24
OC1B' 20 28 59 2F                    msg_yn:      db    ' (Y/N)? $'
OC1F' 4E 29 3F 20
OC23' 24
OC24' 0D 0A 0D 0A                    msg_rt:      db    cr,lf,cr,lf
OC28' 50 72 65 73                    db    'Press RETURN to Continue $'
OC2C' 73 20 52 45
OC30' 54 55 52 4E
OC34' 20 74 6F 20
OC38' 43 6F 6E 74
OC3C' 69 6E 75 65
OC40' 20 24
OC42' 20 20 28 55                    msg_u0:      db    ' (User 0)$'
OC46' 73 65 72 20
OC4A' 30 29 24

OC4D' 00                            BI_D_FLAG:    db    000H ; DIR/DIRS flag set if files found & not listed

OC4E' 4E 4F 4E 2D                    msg_ns:      db    'NON-'
OC52' 53 59 53 54                    msg_sf:      db    'SYSTEM FILE(S) EXISTS'
OC56' 45 4D 20 46
OC5A' 49 4C 45 28
OC5E' 53 29 20 45
OC62' 58 49 53 54
OC66' 24

OC67' 00                            sav_scb_19:   db    0 ; Bit 1 set for profile, Bit 0 set by OVERLY
OC68' 00 00                          db    0,0

OC6A' 3A                            SCB_3A:      db    03AH ; Offset within SCB

```

```

0C6B' 00                                db      000H    ; Get

0C6C'                                pfcbl52:          ; Parsed File Control Block
0C6C' 0000                            ptr_line:        DW      00000H    ; -> first character in CCP command line
0C6E' 0C00'                          DW      P_FCB    ; -> target FCB

0C70' 00                                user:          db      000H
0C71' 00                                disk:          db      000H
0C72' 00                                use_disk:       DB      000H
0C73'                                tmp_disk:      db
0C73' 01                                sub_fcb:       db      001H
0C74' 24 24 24 20                      db      'SS$    SUB'
0C78' 20 20 20 20
0C7C' 53 55 42
0C7E' 00                                db      000H

; remainder continues beyond end of file

0C96'                                end_fcb        equ      sub_fcb+35
0C97'                                mod_scb_1A     EQU      end_fcb + 1    ; (screen width+1)/16
0C98'                                sav_scb_1C     EQU      end_fcb + 2    ; max lines on screen - 1
0C99'                                y_curs        equ      end_fcb + 3    ; current line number
0C9A'                                sav_scb_2C     equ      end_fcb + 4    ; PAGEing flag
0C9B'                                ptr_tok        equ      end_fcb + 5    ; Pointer to token in command line
0C9D'                                ptr_bi        equ      end_fcb + 7
0C9F'                                bi_rec_ptr     equ      end_fcb + 9    ; Pointer in record used by BI_TYPE
0CA0'                                bdos_pg       equ      end_fcb + 10
0CA1'                                flag_5B       equ      end_fcb + 11    ; Set > 0 when I detected
0CA2'                                psword        equ      end_fcb + 12    ; 8+2 bytes of password
0CAC'                                fcb1usr      equ      end_fcb + 22    ; User + 1, 0 = use default
0CAD'                                fcb1         equ      end_fcb + 23    ; Drive + 1, 0 = use default
0CAE'                                fcb1_n       equ      end_fcb + 24    ; Filename
0CB6'                                fcb1_t       equ      end_fcb + 32    ; Filetype
0CCE'                                tp_base      equ      end_fcb + 56    ; OVERLY sets this to 0100H
0CD0'                                P_FCB        equ      end_fcb + 58    ; target FCB (36 bytes)

0CF4'                                c_buff       equ      end_fcb + 94    ; Max EAH = 1+1+E7+1 bytes

0DE0'                                stack        equ      end_fcb + 94 + 236

0E2D'                                Top_sp       equ      end_fcb + 94 + 236 + 77

```

END

## Macros:

## Symbols:

0B80'	A2HL	0B71'	ASC2BIN	083B'	A_OUT
0077'	BAD_LOAD	0005'	BDOCS	08F0'	BDOCS_DOL
0CA0'	BDOCS_PG	08EB'	BDOCS_TST	0294'	BITMAP
066C'	BI_BDOCS	0575'	BI_DIR	057D'	BI_DRS
0582'	BI_DS	0599'	BI_DS1	05B9'	BI_DS2
05CE'	BI_DS3	05E5'	BI_DS4	0C4D'	BI_D_FLAG
0622'	BI_EPA	064C'	BI_EPA1	055A'	BI_HROC
08E2'	BI_READ	0C9F'	BI_REC_PTR	0651'	BI_REN
0692'	BI_REN_DSK	05F4'	BI_TYP	0605'	BI_TYP1
0615'	BI_USR	0000'	CCP	0537'	CCP_BI
0941'	CCP_NS	040C'	CCP_PROMPT	09A8'	CCP_RSX
091D'	CHR_OK	0AF1'	CLR_SCB	0AAE'	CLR_SCB_18
08BF'	CMP_SET	0BC2'	CMP_SET1	0BC4'	CMP_SET2
0BD0'	CMP_SET3	0BED'	CMP_SET4	0BE2'	CMP_SET5
050A'	COLON	0A39'	COMMAND	048B'	COMMAND_LINE
0496'	COMMAND_LINE1	04E8'	COMMAND_TYPE	0763'	COM_SUB_PRL
086D'	CONIN	083C'	CONOUT	0866'	CONSOLE_ABORT
0AA7'	COPY2SCB	0AAE'	COPY_C	03D9'	COPY_CMD
0AB7'	COPY_TAIL	000D'	CR	0B09'	CRLF
0B08'	CRLF_B	03C4'	CR_DISK	03BA'	CR_USR
0A49'	C_1ST4	0CF4'	C_BUFF	0A6B'	C_COL1
0A85'	C_COL2	0A9C'	C_COL3	0A67'	C_COLON
0A5E'	C_NAME	0A58'	C_NOT4	0A53'	C_NXT4
0C71'	DISK	01F7'	DONE_PRL	0B25'	DSP_DIG
0B1A'	DSP_TEN	0B13'	DSP_USR	00F0'	ENABLE_RSX
0C96'	END_FCB	09CB'	ERA_CMD_RSX	0100'	ERA_RSX
06B8'	ERR_FN	006A'	ERR_LOAD	06BB'	ERR_MSG
0CAD'	FCB1	0CAC'	FCB1USR	0CAE'	FCB1_N
0CB6'	FCB1_T	0891'	FDOS	0BB5'	FTILL_3F
0CA1'	FLAG_5B	06C1'	FN_SUB	084E'	GET_CON_BUFF
09D8'	GET_DEF_FCB	09DB'	GET_FCB	0AFF'	GET_SCB
06AB'	GET_TAIL	0504'	GO_COMMAND	0686'	GO_XBI
00D0'	INIT_RSX	0758'	JP_HL_A	0226'	LDIR
00FB'	LD_SCB	0064'	LENSCB	000A'	LF
0839'	LF_OUT	0B05'	LINEDOUT	00EB'	LINK_RSX
0170'	LOAD1	0176'	LOAD2	018F'	LOAD3
01FA'	LOAD4	0006'	LOADER	009E'	LOAD_GENCOM
001B'	LOAD_MAIN	0154'	LOAD_MORE	0130'	LOAD_OVLY
0082'	LOAD_RSX	02BE'	LOAD_SP	029E'	LOAD_STACK
06EA'	LOAD_TP	0880'	LOGIN_DSK	00BF'	L_GEN1
0092'	L_RSK1	009D'	L_RSK2	0708'	L_TP1
071B'	L_TP2	0720'	L_TP3	0721'	L_TP4
0754'	L_TP5	0776'	L_TP6	0067'	MEM_ERR
0C97'	MOD_SCB_1A	0253'	MSG_BL	0C14'	MSG_ER
0C02'	MSG_NF	0CAE'	MSG_NS	0C0A'	MSG_RQ
0C24'	MSG_RT	0C52'	MSG_SF	0C42'	MSG_U0
0BF3'	MSG_US	0C1B'	MSG_YN	023B'	MS_READ
0959'	MULTIPLE	0972'	M_RSK1	097D'	M_RSK2
0525'	NEW_DSK	051A'	NEW_USR	092A'	NOT_5B
0A10'	NOT_FS	0937'	NOT_LC	0406'	NO_CHAIN
0459'	NO_SUB	0A28'	NULL_TOK	0BB7'	NXT_3F
0921'	NXT_CCP	0AB9'	NXT_COPY	0B73'	NXT_NO

0944'	NXT_NS	0104'	NXT_RSX	06A0'	NXT_WC
0009'	NX_RSX	088B'	OPEN_FCB1	0294'	OS_BASE
080D'	OVERL1	0789'	OVERLY	01CA'	PAGE_RELOC
09FE'	PARSE1	0A09'	PARSE2	0A31'	PARSE3
0A38'	PARSE4	09EA'	PARSE_FCB	030A'	PATCH
0C6C'	PFCB152	029C'	PHYS_ERR	01D9'	PRL_BIT
01EA'	PRL_BIT1	01F3'	PRL_BIT2	03FC'	PROF_S
0841'	PROMPT	000C'	PR_RSX	0CA2'	PSWORD
0C8D'	PTR_BI	0C6C'	PTR_LINE	028D'	PTR_SCB
0C9B'	PTR_TOK	0AF9'	PUT_SCB	0233'	PUT_SCB_4A
0CD0'	P_FCB	0406'	QUIT	044F'	QUIT_SUB
0358'	RDY59	005F'	RET_OK	0298'	SAVE_DE
029A'	SAVE_SP	0C67'	SAV_SCB_19	0C98'	SAV_SCB_1C
0C9A'	SAV_SCB_2C	028F'	SAV_SCB_5E	08F6'	SCAN_OCP
009C'	SCB	00A1'	SCB_05	00AB'	SCB_0F
00AC'	SCB_10	00AE'	SCB_12	00AF'	SCB_13
00B0'	SCB_14	00B1'	SCB_15	00B3'	SCB_17
00B4'	SCB_18	00B5'	SCB_19	00B6'	SCB_1A
00B8'	SCB_1C	00BA'	SCB_1E	00BC'	SCB_20
00C8'	SCB_2C	00C9'	SCB_2D	00CF'	SCB_33
00D3'	SCB_37	0C6A'	SCB_3A	0290'	SCB_3C_PB
00DA'	SCB_3E	00ED'	SCB_44	00E6'	SCB_4A
00E7'	SCB_4B	00EB'	SCB_4C	00EC'	SCB_50
00F9'	SCB_5D	00F9'	SCB_62	0292'	SCB_62_PB
08CA'	SEARCH_FCB	08C3'	SEARCH_FIRST	08C8'	SEARCH_NEXT
077A'	SEARCH_TYPE	00F8'	SET_62	0080'	SET_CMD
0878'	SET_DEF_DMA	087B'	SET_DMA	0050'	SET_DR
0004'	SET_DSK	006C'	SET_F2	005C'	SET_FCB
0051'	SET_PA1	0054'	SET_PA2	0053'	SET_PL1
0056'	SET_PL2	0AE4'	SET_SCB_18	0231'	SET_SCB_4A
0886'	SET_USR	0464'	SHOW_PROMPT	0955'	SKIP_B
091C'	SKIP_C	0DE0'	STACK	031A'	START
0849'	STROUT	06CD'	SUBMIT	0C73'	SUB_FCB
03E7'	TEST_19	0C73'	TMP_DSK	020F'	TOP_LESS_B
0296'	TOP_OVLY	0E2D'	TOP_SP	0100'	TPA
0CCE'	TP_BASE	0B36'	TST_CMD_END	0B3A'	TST_QUIT
0ADD'	TST_SCB	0ADA'	TST_SCB_18	069E'	TST_WILDCARD
0AC4'	TYPE_CHAR	0770'	TYPE_PROC	04F9'	TYPE_SEARCH
0C70'	USER	0C72'	USE_DSK	0B52'	VALIDATE_USR
0B2A'	VDUMSG	0BA6'	VDU_A	0BA4'	VDU_HL
0566'	VDU_CMD_DSK	056D'	VDU_CR_DSK	0B98'	VDU_DE
0569'	VDU_DSK	0570'	VDU_DSK1	0B8D'	VDU_FN
0816'	VDU_OUT	0000'	WBOOT	0090'	XSCB
0090'	XSCB00	0091'	XSCB01	0092'	XSCB02
0093'	XSCB03	0098'	XSCB08	0C99'	Y_CURS

No Fatal error(s)



## INDEX

# The Amstrad CP/M Plus - Index

Abstractions .....	17
Access .....	34
AL0/1 .....	186
Algol .....	115
Algol/M .....	330
ASCOM .....	13, 112
ASEG .....	373, 379
ASM.COM .....	325, 374
ASMX .....	377
Assembler Pseudo-Ops .....	384
Astec C .....	323, 324
AUX: .....	55, 71, 55
AUXIN: .....	55, 99
AUXOUT: .....	55, 99, 270
AXI: .....	71
AXO: .....	71
BASCOM .....	11, 353
BASIC language .....	114
BASIC Compiler Error Messages .....	353
BASIC-E .....	12, 330
BAUDRATE .....	54, 103, 252
BCB .....	255
BDOS calling conventions .....	143
BDOS character functions .....	147, 152
BDOS clock functions .....	223
BDOS Disk FCB Initialisation and return codes .....	218
BDOS disk functions .....	160
BDOS Drive Functions .....	164
BDOS FCB and Directory Functions .....	188
BDOS FUNCTION 0 System Reset .....	141, 234
BDOS FUNCTION 1 Console Input .....	152, 247
BDOS FUNCTION 2 Console Output .....	152
BDOS FUNCTION 3 Auxiliary input .....	152
BDOS FUNCTION 4 Auxiliary output .....	153
BDOS FUNCTION 5 List output .....	153
BDOS FUNCTION 6 Direct console input/output .....	151, 153, 247
BDOS FUNCTION 7 Auxiliary input status.....	154, 247
BDOS FUNCTION 8 Auxiliary output status .....	154
BDOS FUNCTION 9 List output string .....	150, 154
BDOS FUNCTION 10 Console buffer input .....	149, 155, 156, 247
BDOS FUNCTION 11 Console Input Status .....	148, 157
BDOS FUNCTION 12 Return Version Number .....	234
BDOS FUNCTION 13 Reset disk system .....	164, 166, 247
BDOS FUNCTION 14 Select Disk .....	164, 166
BDOS FUNCTION 15 Open File FCB .....	164, 172, 188, 247
BDOS FUNCTION 16 Close File FCB.....	173, 189, 247
BDOS FUNCTION 17 Search first FCB.....	190
BDOS FUNCTION 18 Search next FCB .....	184, 190
BDOS FUNCTION 19 Delete File FCB .....	173, 191
BDOS FUNCTION 20 Read Sequential .....	173, 192, 247
BDOS FUNCTION 21 Write Sequential .....	173, 193
BDOS FUNCTION 22 Make File FCB .....	171, 194, 248
BDOS FUNCTION 23 Rename File .....	195
BDOS FUNCTION 24 Return Login vector .....	167
BDOS FUNCTION 25 Return Current Disk .....	164, 167
BDOS FUNCTION 26 Set DMA transfer address .....	172, 196, 248

# The Amstrad CP/M Plus - Index

BDOS FUNCTION 27	Get ALLOC vector .....	197, 247
BDOS FUNCTION 28	Write Protect disk .....	167
BDOS FUNCTION 29	Get R/O vector .....	168, 248
BDOS FUNCTION 30	Set FCB file attributes .....	198
BDOS FUNCTION 31	Get DPB parameter block .....	199, 248
BDOS FUNCTION 32	Set/Get User number .....	164, 168, 248
BDOS FUNCTION 33	Read Random .....	175, 200
BDOS FUNCTION 34	Write Random .....	176, 201
BDOS FUNCTION 35	Compute File Size .....	175, 202, 248
BDOS FUNCTION 36	Set Random Record .....	175, 203
BDOS FUNCTION 37	Reset Drive .....	164, 169
BDOS FUNCTION 38	Access Drive (MP/M) .....	169
BDOS FUNCTION 39	Free Drive (MP/M) .....	170
BDOS FUNCTION 40	Write Random with zero fill .....	176, 204
BDOS FUNCTION 44	Set Multi-Sector Count .....	172, 205
BDOS FUNCTION 45	Set BDOS disk error mode .....	171, 206
BDOS FUNCTION 46	Get Disk Free Space .....	207
BDOS FUNCTION 47	Chain to program .....	141, 237
BDOS FUNCTION 48	Flush Buffers .....	208
BDOS FUNCTION 49	Get/Set System Control Block .....	142, 232
BDOS FUNCTION 50	Direct BIOS calls .....	142, 238
BDOS FUNCTION 59	Load Overlay .....	141, 240
BDOS FUNCTION 60	Direct RSX calls .....	242
BDOS FUNCTION 98	Free Temporary Datablocks .....	209
BDOS FUNCTION 99	Truncate File FCB .....	176, 210
BDOS FUNCTION 100	Set Directory Label .....	211
BDOS FUNCTION 101	Return Directory Label .....	212
BDOS FUNCTION 102	Read file date stamp and Password Mode ...	213
BDOS FUNCTION 103	Write file password XFCB.....	181, 214
BDOS FUNCTION 104	Set Date and Time .....	225
BDOS FUNCTION 105	Get Date and Time .....	225
BDOS FUNCTION 106	Set Default Password .....	215
BDOS FUNCTION 107	Return Serial Number .....	234
BDOS FUNCTION 108	Get/Set Program return code .....	141, 235
BDOS FUNCTION 109	Get/Set Console mode .....	157, 230
BDOS FUNCTION 110	Get/Set output delimiter .....	150, 158
BDOS FUNCTION 111	Output string to CONSOLE device .....	150, 158
BDOS FUNCTION 112	Output string to LIST device .....	150, 159
BDOS FUNCTION 152	Parse Filename .....	171
BDOS system and miscellaneous FUNCTIONS	.....	233
BDOS System Control Block .....		226
BDS C .....		327
BIOS Calling conventions .....		249
BIOS character FUNCTIONS .....		251
BIOS disk FUNCTIONS .....		254
BIOS Jump Vector .....		249
BIOS system FUNCTIONS .....		257
BSH .....		186
BSTAM .....		13, 112
Buffer Control Block Fields .....		255
Bulletin board .....		111
C language.....		114
CB-80 .....		12
CBASIC .....		12, 334
CBASIC Compiler Toggles .....		334
CBASIC Reserved Words .....		342
CBASIC Predefined Functions .....		343

# The Amstrad CP/M Plus - Index

CCP.COM .....	244
Changing drives .....	25
Character control block .....	159
Character string i/o .....	148
Character substitution and control .....	151
Clock .....	34
COMMON .....	373
CON: .....	55, 71
Concurrent CP/M .....	8
CONIN: .....	55, 99
CONOUT: .....	55, 99
Console Buffer Block .....	155
CONSOLE: .....	55
CONST .....	266, 271
Control-C .....	18
COPYSYS .....	52
CP/M 1.4 .....	7
CP/M 2.2 BDOS compatibility .....	243
CP/M 2.2 Utility Compatibility .....	245
CP/M early development .....	6
CP/M user group (UK) library .....	373, 377
CP/M User Group (UK) .....	373
CP/M- why it is so popular .....	5
CP/M-86 .....	7
CREF.COM .....	376, 383
CRT .....	99
CSEG .....	373, 379
Date .....	28, 34, 53
Date stamp .....	66, 180
DB .....	377
DBASE-II .....	12
DD .....	332
DDT.COM .....	374
DEVICE .....	54
DEVICE.COM .....	103
DIR .....	24, 29, 35, 45, 56
Direct Console i/o .....	151
Directory Access .....	179
Directory FCB .....	160
Directory Scan .....	185, 29, 35, 45, 56
Directory Search .....	183
DIRSYS .....	45
Disckit .....	28
Disk Parameter Block .....	255
Disk Parameter Header .....	254
DISKSTAT .....	186
DPB .....	182, 186, 255
DPH .....	186, 254
Dr Kildall .....	374
DR-GRAPH .....	13
DRM .....	186
DS .....	377
DSEG .....	373, 379
DSM .....	186
DUMP .....	59
DW .....	377

ED .....	31, 84
ELSE .....	377
END .....	377
ENDIF .....	377
ENDM .....	378
EOF: .....	71
EGU .....	378
ERA .....	24, 34
ERASE .....	43
Example of Direct Bios Interface.....	258
EXCOM .....	65
EXITM .....	378
EXM .....	182, 186
Extended intel mnemonics.....	374
Extended intel Z80 mnemonics .....	377
Extended SCB .....	227
EXTERN .....	379
EXTRN .....	373
Filename .....	18
FORTH language .....	115
FORTRAN language .....	116
Gary Kildall .....	375
GDOS FUNCTIONS .....	296
GDOS OPCODE 1 - Open Workstation .....	299
GDOS OPCODE 2 - Close Workstation .....	300
GDOS OPCODE 3 - Clear Workstation .....	301
GDOS OPCODE 4 - Update Workstation .....	301
GDOS OPCODE 5 - Perform device-specific operation .....	302
GDOS OPCODE 6 - Output a polyline .....	303
GDOS OPCODE 7 - Output polymarkers .....	304
GDOS OPCODE 8 - Output Graphics Text at XY coordinates .....	304
GDOS OPCODE 9 - Output polyfilled area (polygon) .....	305
GDOS OPCODE 10 - Display Cell Array .....	306
GDOS OPCODE 11 - Output a Generallised Drawing Primitive ...	307
GDOS OPCODE 12 - Set Character Height .....	308
GDOS OPCODE 13 - Set Text Direction .....	309
GDOS OPCODE 14 - Specify colour index value .....	309
GDOS OPCODE 15 - Set Polyline linetype .....	310
GDOS OPCODE 16 - Set Polyline line Width .....	310
GDOS OPCODE 17 - Set Polyline colour index .....	311
GDOS OPCODE 18 - Set Polymarker type .....	311
GDOS OPCODE 19 - Set Polymarker height (scale) .....	312
GDOS OPCODE 20 - Set Polymarker colour index .....	312
GDOS OPCODE 21 - Set The hardware text font .....	313
GDOS OPCODE 22 - Set The Colour Index .....	313
GDOS OPCODE 23 - Set The Interior Fill Style .....	314
GDOS OPCODE 24 - Set The Fill Style Index .....	314
GDOS OPCODE 25 - Set The Fill colour Index .....	315
GDOS OPCODE 26 - Return colour representation .....	315
GDOS OPCODE 27 - Return cell array definition .....	316
GDOS OPCODE 28 - Return locator position .....	317
GDOS OPCODE 29 - Return value of valuator device .....	318
GDOS OPCODE 30 - Return choice device status keys .....	319
GDOS OPCODE 31 - Return string from specified string .....	320
GDOS OPCODE 32 - Set writing mode .....	321

# The Amstrad CP/M Plus - Index

GDOS OPCODE 33 - Set input mode .....	322
GENCOM .....	60, 241, 271
GET .....	61
GSS-GRAPH .....	13
GSS-KERNEL .....	273
GSS-PLOT .....	273
GSX .....	9, 13, 264, 272
GSX Alpha Text Functions .....	298
GSX Cell Array functions .....	298
GSX Control functions .....	297
GSX Filled Area Functions .....	297
GSX Graphic input functions .....	298
GSX Graphic Text Functions .....	298
GSX Line Drawing Functions .....	297
GSX Marker Drawing Functions .....	297
GSX Mode functions .....	297
HELP .....	63
How to copy a disk .....	28
How to copy a file .....	33
How to create a file .....	31
How to datestamp a file .....	34
How to edit a file .....	32
How to erase a file .....	34
How to find whats on disk .....	29
How to format a disk .....	27
How to initialise a disk for date stamping .....	28
How to name a file .....	29
How to print a file .....	30
How to rename a file .....	34
How to run a program .....	35
How to see what room is on disk .....	29
IF .....	378
INITDIR .....	28, 34, 66
INP: .....	71
Intel Assembler Pseudoops .....	377
Interfacing to other devices .....	106
Introduction to RSXs .....	264
IRP .....	378
IRPC .....	378
Jargon - or shorthand .....	17
KBASIC .....	12
KERMIT .....	112
KEYBOARD .....	55, 99
KSAM .....	12
L80.COM .....	376, 381
LIB .....	87
LIB-80 .....	383
LIB.COM .....	375, 376
LINK.COM .....	89, 144, 375
LINKASM .....	377
LISP .....	115
LOAD.COM .....	374
LOCAL .....	378

# The Amstrad CP/M Plus - Index

Location of CCP .....	244
Logical and physical devices .....	99
Logical devices .....	102
LOGO .....	115
LOTUS 123 .....	13
LPT .....	99
LST: .....	55, 71, 99
M80 the Microsoft assembler.....	144, 376, 380
MAC .....	91, 374
MACASM .....	377
Macro .....	91, 378
Macro Directives .....	378
MACROII.COM .....	375
MBASIC .....	11
MBASIC (Microsoft BASIC) Interpreter .....	346
Microsoft linker .....	381
MILMON80 .....	377
ML-80 .....	376
MOVE-IT .....	13, 112
Multiplan .....	13
NAME .....	373, 379
Naval Postgraduate School, Monterey .....	374
[NO PAGE] .....	31
Normalized Device coordinates .....	273, 283, 285
NUL: .....	71
Once round the block .....	27
ORG .....	378
OUT: .....	71
Pascal .....	114, 330
PASCAL MT+ .....	354
Pascal/MT Reserved Words .....	357
Pascal/MT Symbolic Debugger .....	356
PASM .....	375, 385
PATCH .....	67
Physical devices .....	99
PIP .....	28, 31, 68
PLI .....	115
PLINKII.COM .....	376
PLINK.COM .....	376
PRINT USING statements .....	349
PRINTER: .....	55
Printing a file on paper .....	31
PRN: .....	71
PROFILE.SUB .....	35
Programmers nostrums .....	128
Programming with GSX .....	282
ProPascal .....	114
Protocol .....	103
PUBLIC .....	373, 379
PUT .....	74
Random Access .....	174
Random access using sequential disk functions .....	177
RATFOR .....	116

# The Amstrad CP/M Plus - Index

Real time clock .....	34
REN .....	24, 34
RENAME .....	47
REPT .....	378
Restrictions on Z80 instructions and registers .....	140
RMAC .....	95, 375
RS232 Interface .....	103
RSX .....	60, 135, 242, 264
RSX function 60 .....	266
RSX Prefix .....	264
RTMASM .....	377
SAVE .....	76
Scrolling .....	26
Sequential File Access .....	171
Serial and parallel .....	100
SET .....	34, 42, 77, 378
SETDEF .....	79
SETSIO.COM .....	103
SFCB Format .....	180
SHOW .....	29, 80
Shutting down .....	23
SID .....	96, 144, 374, 375
Single character i/o .....	148
SIO .....	99, 100, 101
Small C .....	325
Special BDOS functions .....	186
Starting work .....	23
STKLN .....	379
STOIC .....	115
Stop bits .....	103
SUBMIT .....	81, 327
SuperCalc .....	13
SXMAC .....	258
SYS .....	42
System attribute .....	42
TDL mnemonics .....	377
TDL-Xitan Ops .....	386
Transient command .....	24
Transient Utility Commands .....	51
TurboPascal .....	114
Turtle Graphics .....	115
TYPE .....	24, 30, 31
UKM7 .....	13, 112
Update .....	34
Useful ambiguity .....	18
USER .....	24
User areas .....	40
Using a Relocating Macroassembler .....	359
Using GSX .....	280
VISICALC .....	12
WBOOT .....	253, 257
What a Computer is - and does .....	15
Wildcard .....	18



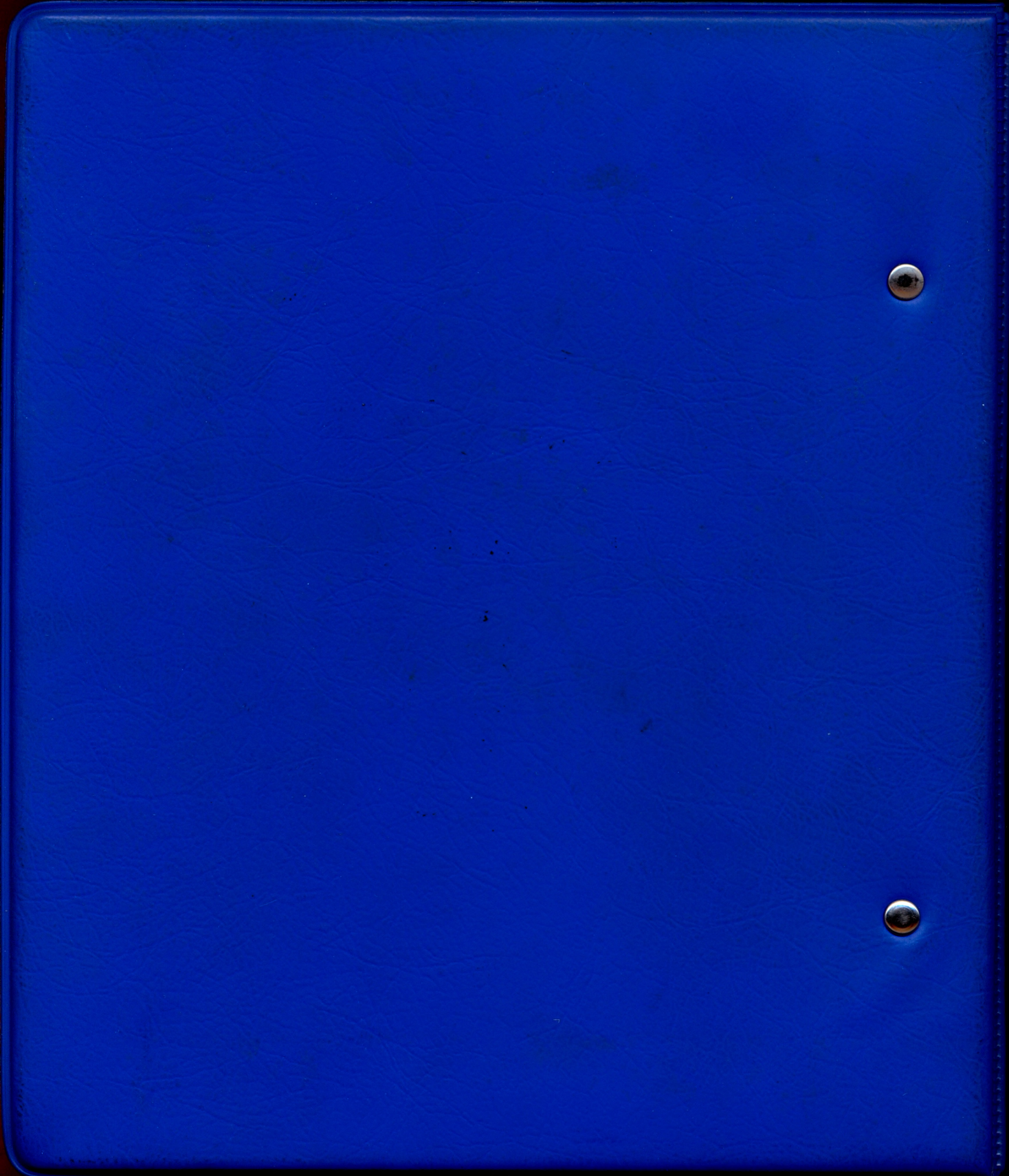
# The Amstrad CP/M Plus - Index

WORDSTAR .....	11
Writing CP/M software .....	113
X-on/X-off .....	103
XFCB Extended FCB .....	181
XMODEM .....	112
XON/XOFF .....	54, 111
XREF .....	98, 344, 375
Z80.LIB .....	373
Z80ASM .....	377
Zilog mnemonics .....	374
ZSID.COM .....	144, 375

# Notes







AMSTRAD

CP/M +

Andrew  
Clarke

David  
Powys-Lybbe